Contents

NATIONAL INSTRUMENTS™

FRC

ni.com/first

NATIONAL INSTRUMENTS

ni.com

FRC LabVIEW Mentor Training
Hands-On Seminar

National Instruments

ni.com/first

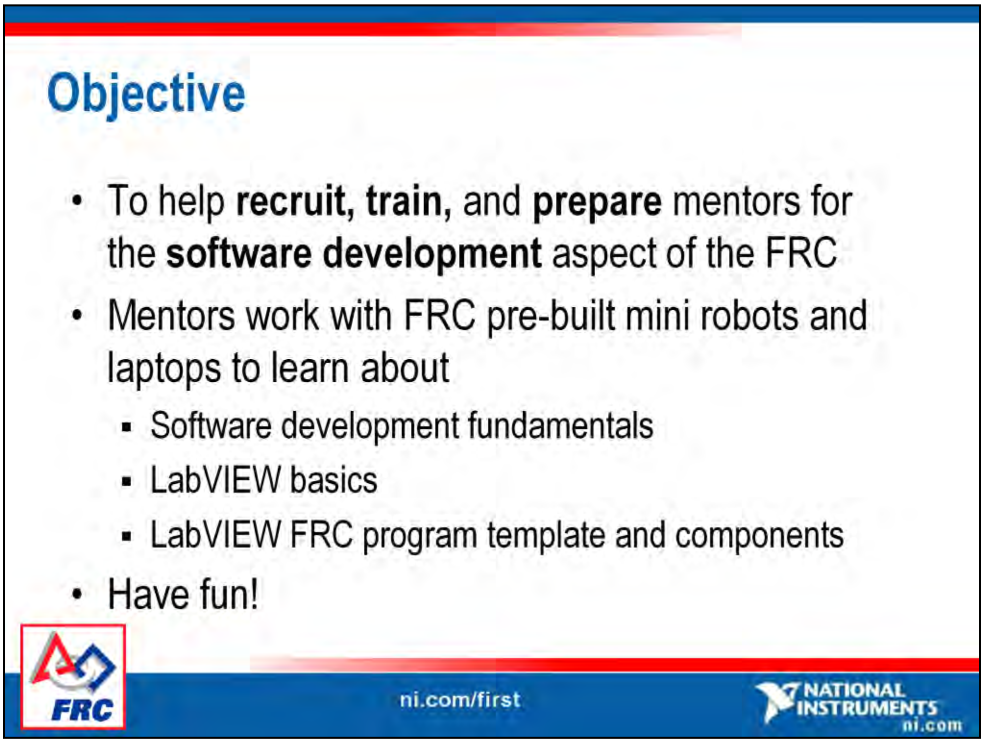
FIRST AND FRC

For **I**nspiration and **R**ecognition of **S**cience and **T**echnology (*FIRST*) is a not-for-profit organization whose mission is to inspire young people to be science and technology leaders by engaging them in exciting mentor-based programs that build science, engineering, and technology skills; that inspire innovation; and that foster well-rounded life capabilities including self-confidence, communication, and leadership.

FIRST **R**obotics **C**ompetition (FRC) combines the excitement of sport with the rigors of science and technology. Under strict rules, limited resources, and time limits, teams of 25 students or more are challenged to raise funds, design a team "brand," hone teamwork skills, and build and program a robot to perform prescribed tasks against a field of competitors. It's as close to "real world" engineering that a student can get.

Grades 9–12 (Ages 14–18)

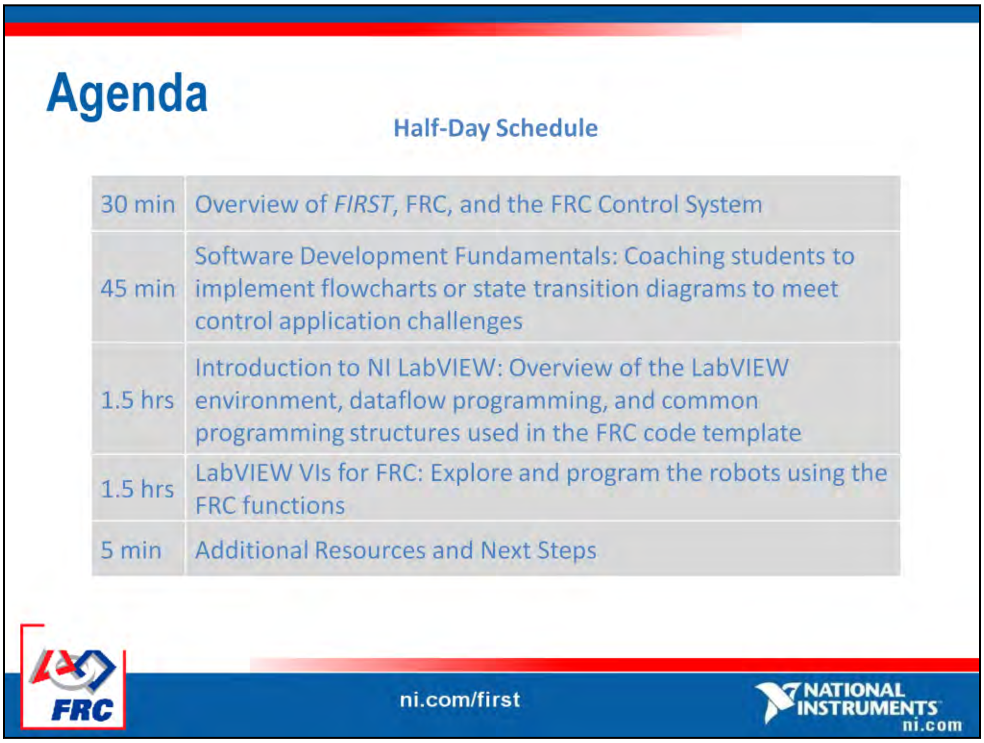
For the 2010/2011 season, 22,475 teams comprising 248,000 students and over 66,000 mentors participated in *FIRST* programs. To learn more about *FIRST* and the *FIRST* family of programs, please visit **usfirst.org**.



Objective

- To help **recruit, train,** and **prepare** mentors for the **software development** aspect of the FRC
- Mentors work with FRC pre-built mini robots and laptops to learn about
  - Software development fundamentals
  - LabVIEW basics
  - LabVIEW FRC program template and components
- Have fun!

ni.com/first



Agenda

Half-Day Schedule

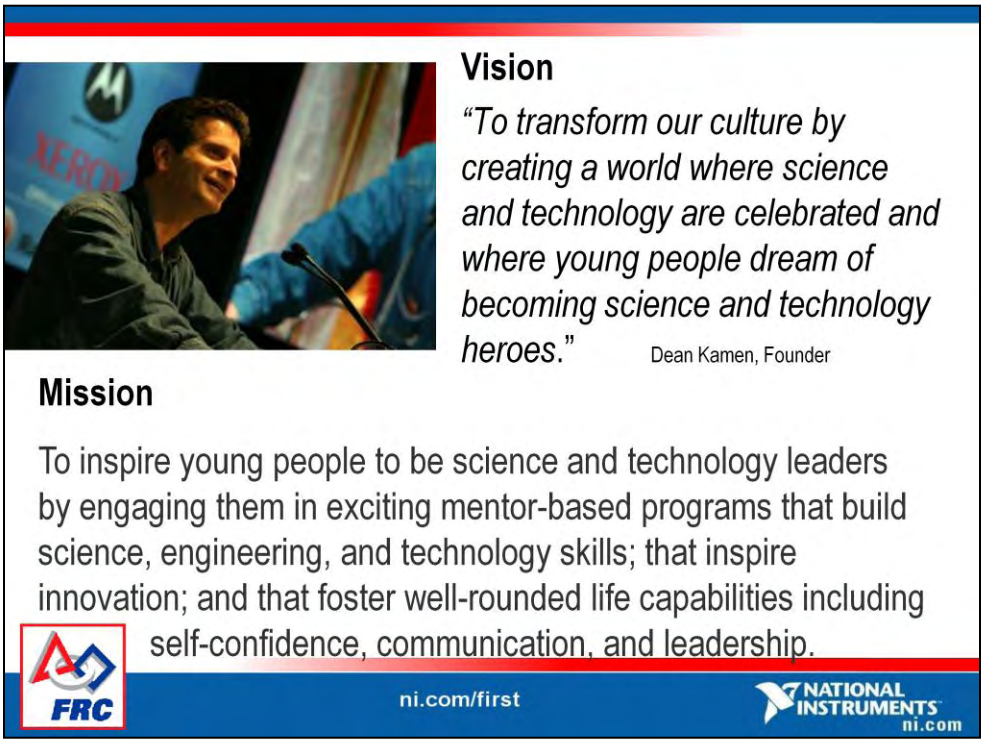
30 min	Overview of *FIRST*, FRC, and the FRC Control System
45 min	Software Development Fundamentals: Coaching students to implement flowcharts or state transition diagrams to meet control application challenges
1.5 hrs	Introduction to NI LabVIEW: Overview of the LabVIEW environment, dataflow programming, and common programming structures used in the FRC code template
1.5 hrs	LabVIEW VIs for FRC: Explore and program the robots using the FRC functions
5 min	Additional Resources and Next Steps

ni.com/first

Please Note: There are more exercises and background information than you can complete in a four-hour period. Mentors are encouraged to take the manual with them so they can review and complete additional optional exercises at their own pace.

Overview *FIRST*, FRC, and FRC Control System

ni.com/first



Vision

"To transform our culture by creating a world where science and technology are celebrated and where young people dream of becoming science and technology heroes." Dean Kamen, Founder

Mission

To inspire young people to be science and technology leaders by engaging them in exciting mentor-based programs that build science, engineering, and technology skills; that inspire innovation; and that foster well-rounded life capabilities including self-confidence, communication, and leadership.

ni.com/first

FIRST AND FRC

For **I**nspiration and **R**ecognition of **S**cience and **T**echnology (*FIRST*) is a not-for-profit organization whose mission is to inspire young people to be science and technology leaders by engaging them in exciting mentor-based programs that build science, engineering, and technology skills; that inspire innovation; and that foster well-rounded life capabilities including self-confidence, communication, and leadership.

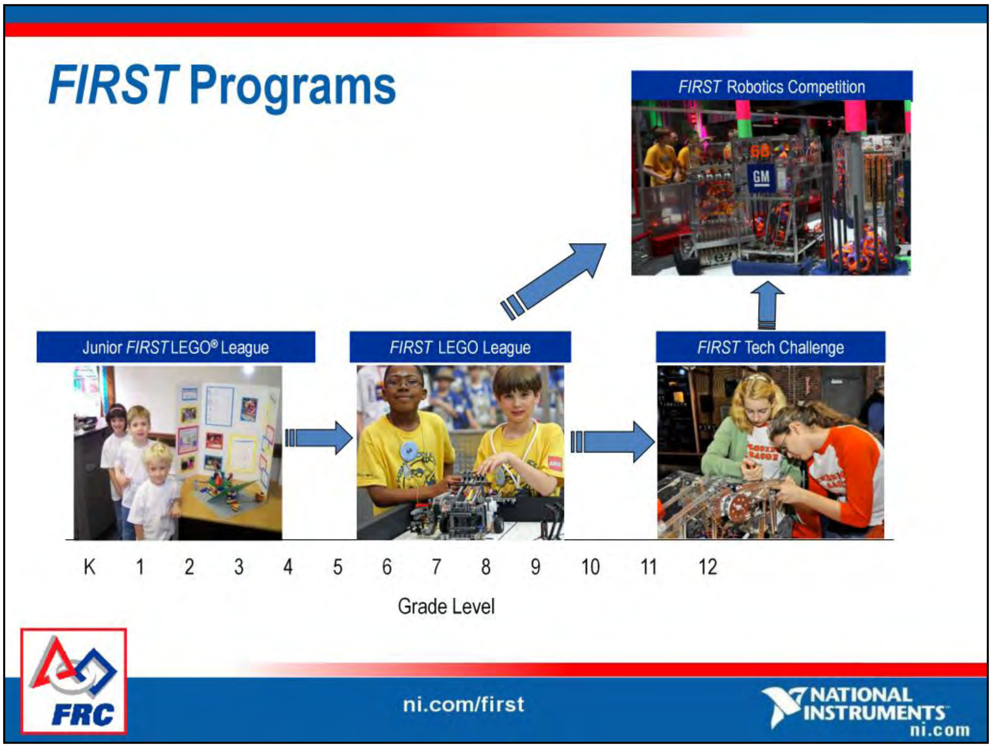
FIRST **R**obotics **C**ompetition (FRC) combines the excitement of sport with the rigors of science and technology. Under strict rules, limited resources, and time limits, teams of 25 students or more are challenged to raise funds, design a team "brand," hone teamwork skills, and build and program a robot to perform prescribed tasks against a field of competitors. It's as close to "real world" engineering that a student can get.

Grades 9–12 (Ages 14–18)

For the 2010/2011 season, 22,475 teams comprising 248,000 students and over 66,000 mentors participated in *FIRST* programs. To learn more about *FIRST* and the *FIRST* family of programs, please visit **usfirst.org**.

Quick Facts

- 501(c)(3) not-for-profit public charity
- Founded 1989, by inventor Dean Kamen
- Headquarters in Manchester, NH
- Board Chairman John Abele, founder chairman of Boston Scientific
- $32 million operating budget
- 3,000+ corporate sponsors
- 92,000 volunteers



FIRST Programs

FIRST Robotics Competition

Junior *FIRST* LEGO® League

FIRST LEGO League

FIRST Tech Challenge

K 1 2 3 4 5 6 7 8 9 10 11 12

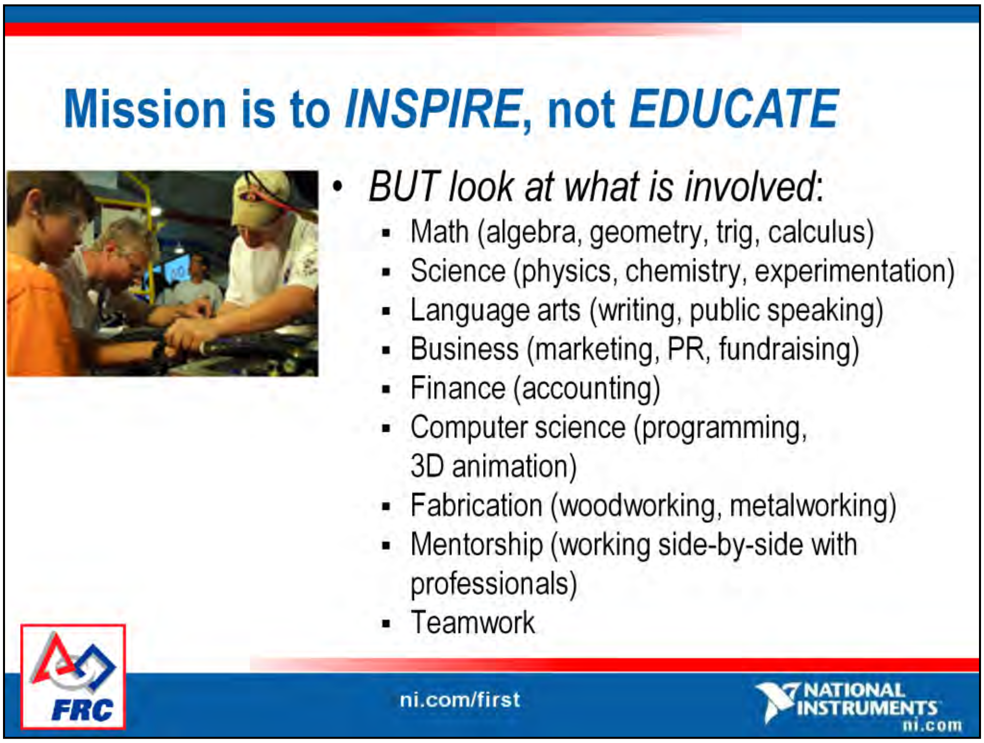
Grade Level

FRC

ni.com/first

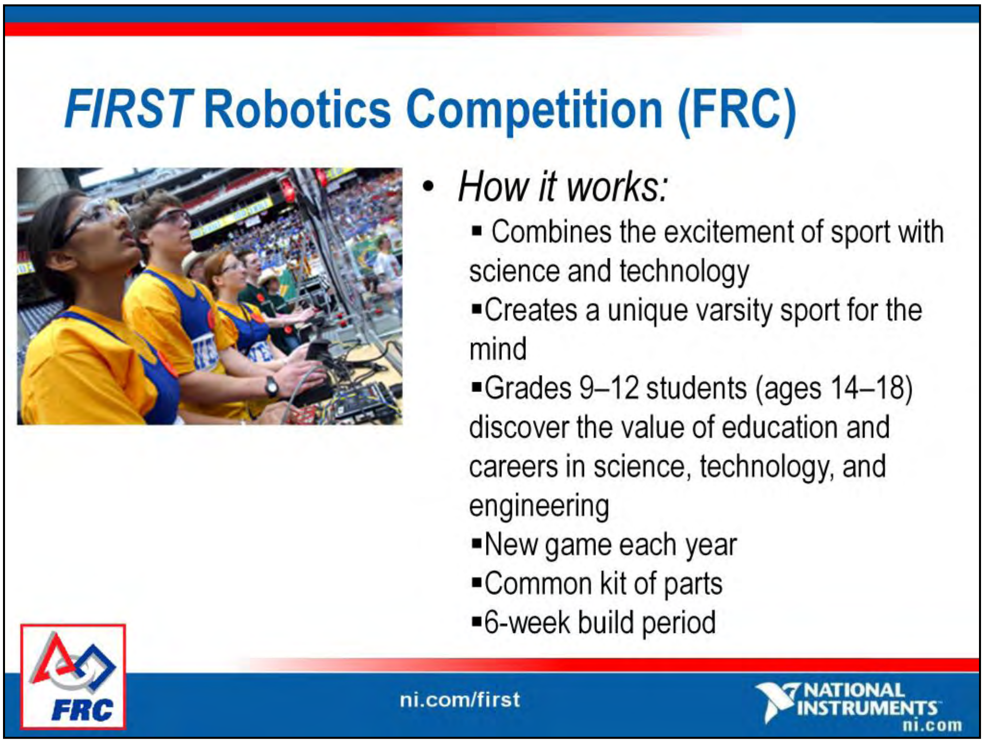
NATIONAL INSTRUMENTS

ni.com



Mission is to *INSPIRE*, not *EDUCATE*

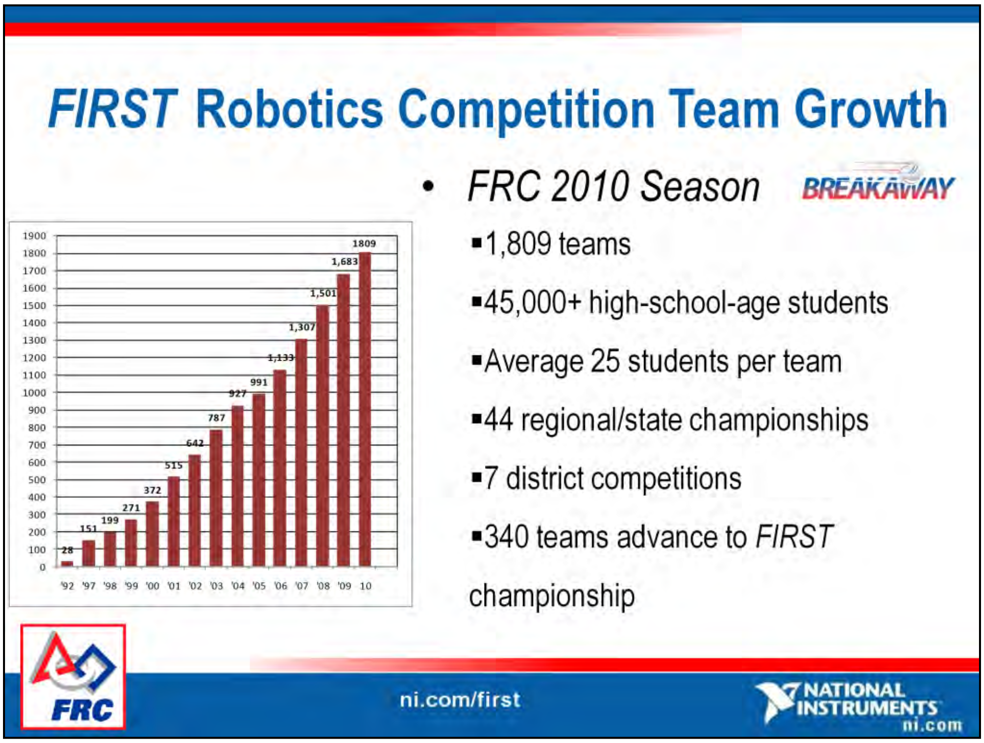
- *BUT look at what is involved*:
  - Math (algebra, geometry, trig, calculus)
  - Science (physics, chemistry, experimentation)
  - Language arts (writing, public speaking)
  - Business (marketing, PR, fundraising)
  - Finance (accounting)
  - Computer science (programming, 3D animation)
  - Fabrication (woodworking, metalworking)
  - Mentorship (working side-by-side with professionals)
  - Teamwork



FIRST Robotics Competition (FRC)

- *How it works:*
  - Combines the excitement of sport with science and technology
  - Creates a unique varsity sport for the mind
  - Grades 9–12 students (ages 14–18) discover the value of education and careers in science, technology, and engineering
  - New game each year
  - Common kit of parts
  - 6-week build period

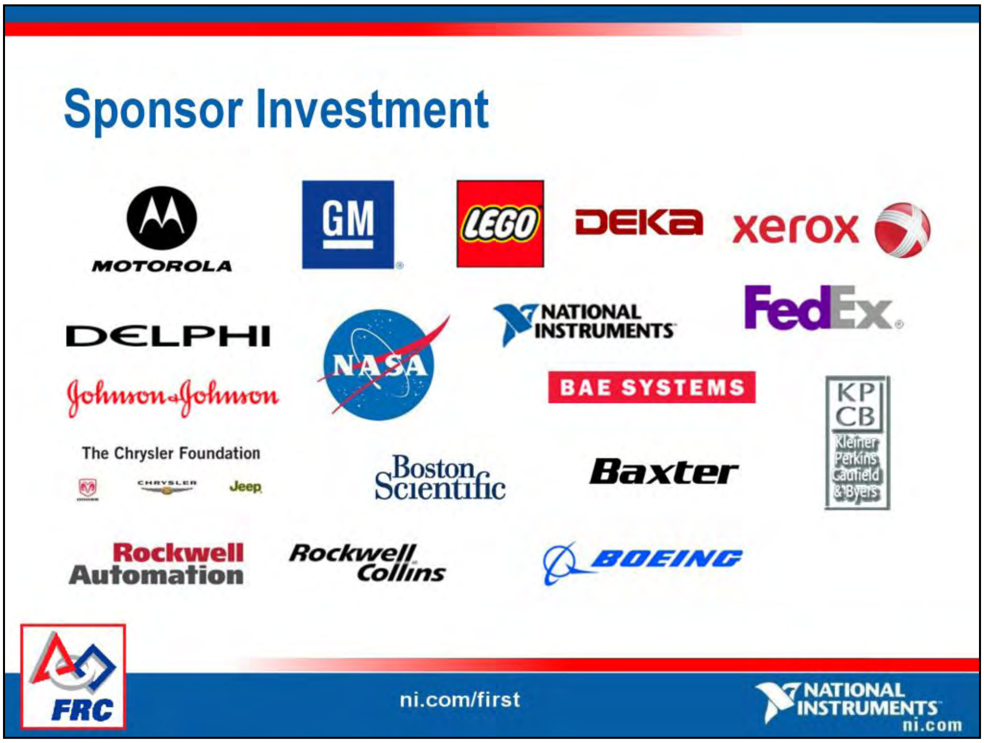
ni.com/first



FIRST Robotics Competition Team Growth

- *FRC 2010 Season* BREAKAWAY
  - 1,809 teams
  - 45,000+ high-school-age students
  - Average 25 students per team
  - 44 regional/state championships
  - 7 district competitions
  - 340 teams advance to *FIRST* championship

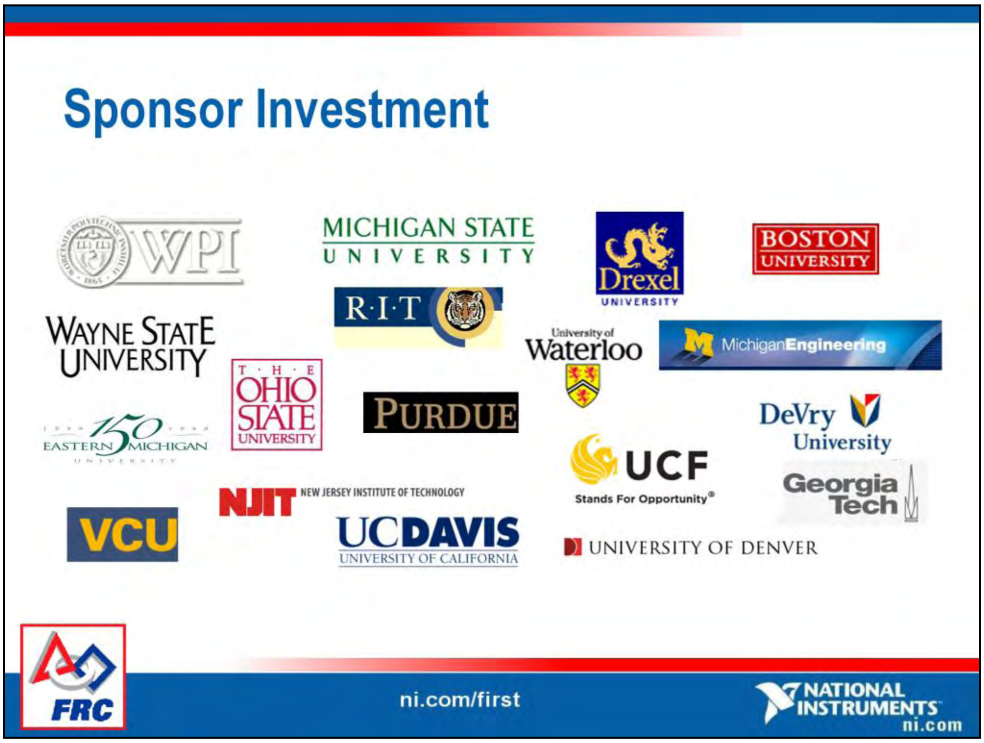
ni.com/first



Sponsor Investment

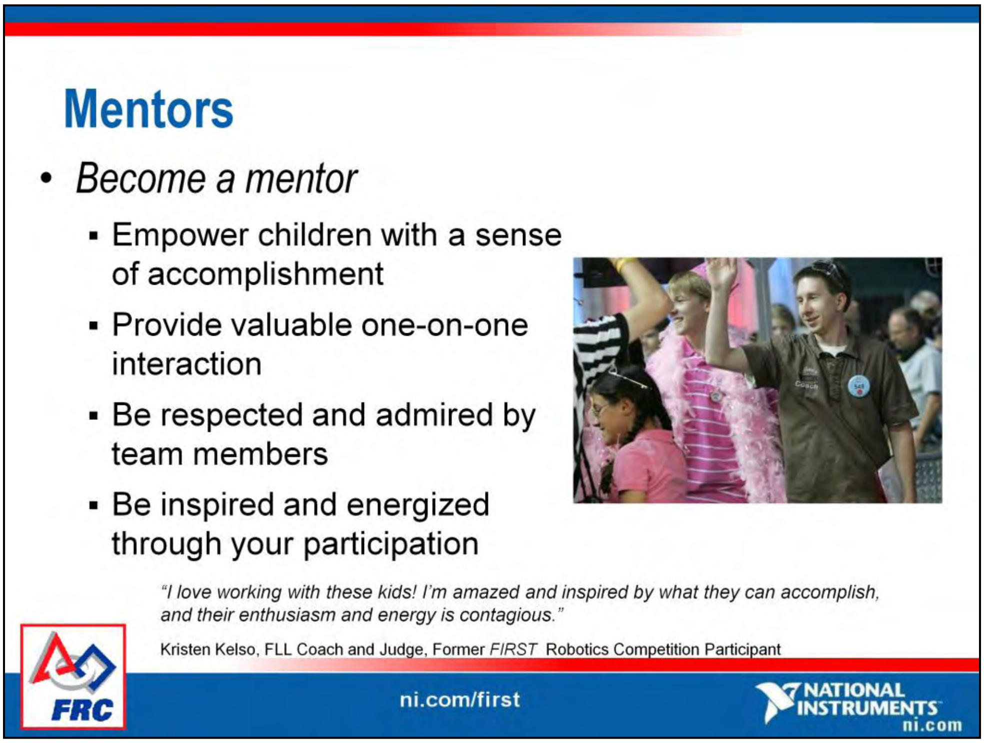
ni.com/first

More than 3,000 leading corporations, foundations, and agencies, including founding sponsors and strategic partners



Sponsor Investment

WPI
MICHIGAN STATE UNIVERSITY
Drexel UNIVERSITY
BOSTON UNIVERSITY
WAYNE STATE UNIVERSITY
R·I·T
University of Waterloo
Michigan Engineering
THE OHIO STATE UNIVERSITY
PURDUE
DeVry University
EASTERN MICHIGAN UNIVERSITY 150
UCF Stands For Opportunity®
Georgia Tech
NJIT NEW JERSEY INSTITUTE OF TECHNOLOGY
VCU
UC DAVIS UNIVERSITY OF CALIFORNIA
UNIVERSITY OF DENVER
FRC
ni.com/first
NATIONAL INSTRUMENTS ni.com



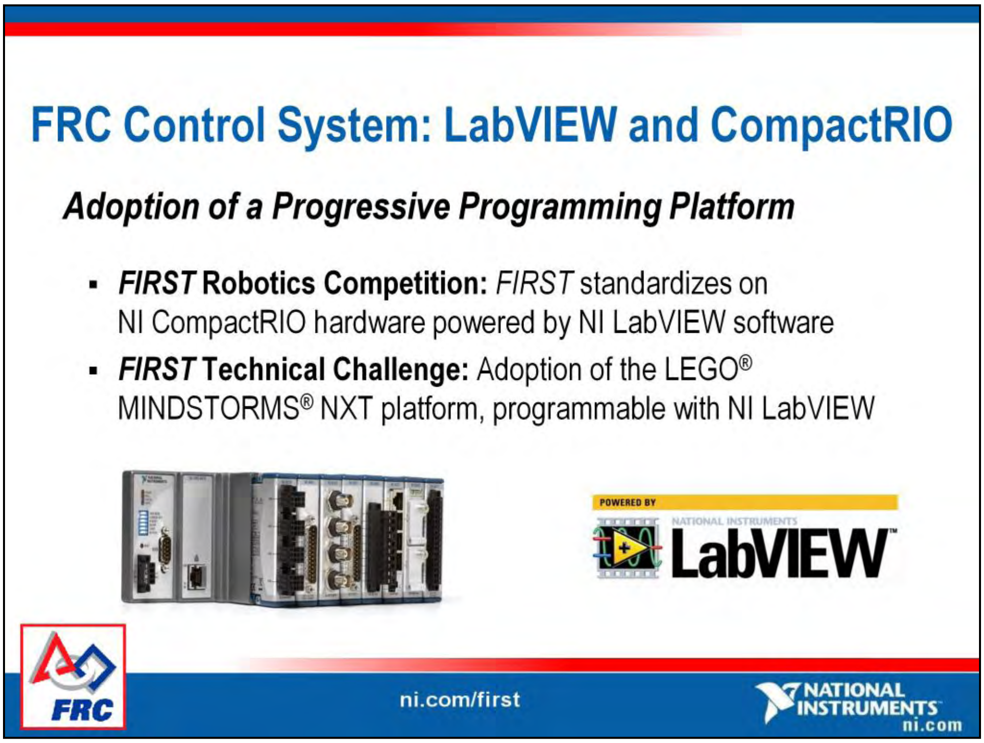
Mentors

- *Become a mentor*
  - Empower children with a sense of accomplishment
  - Provide valuable one-on-one interaction
  - Be respected and admired by team members
  - Be inspired and energized through your participation

"I love working with these kids! I'm amazed and inspired by what they can accomplish, and their enthusiasm and energy is contagious."

Kristen Kelso, FLL Coach and Judge, Former *FIRST* Robotics Competition Participant

http://usfirst.org/roboticsprograms/coachesmentors



FRC Control System: LabVIEW and CompactRIO

Adoption of a Progressive Programming Platform

- ***FIRST* Robotics Competition:** *FIRST* standardizes on NI CompactRIO hardware powered by NI LabVIEW software
- ***FIRST* Technical Challenge:** Adoption of the LEGO® MINDSTORMS® NXT platform, programmable with NI LabVIEW

ni.com/first

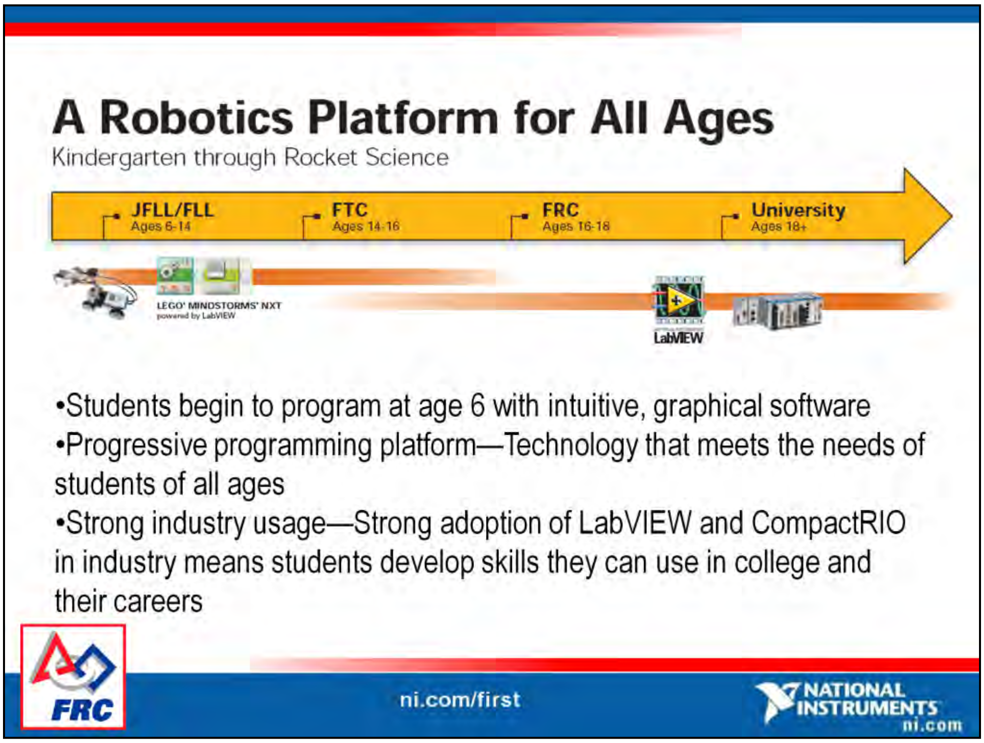
From the 2009 competition, *FIRST* Robotics standardized on LabVIEW and CompactRIO for its next-generation mobile device controller.

Impact of the Controller

•Next-generation control system enables students to build more sophisticated robotics systems and meet more complex challenges

•Hands-on learning helps connect theory and the real world

•Industry-standard technology means students gain knowledge for university studies and professional careers



A Robotics Platform for All Ages

Kindergarten through Rocket Science

JFLL/FLL Ages 6-14

FTC Ages 14-16

FRC Ages 16-18

University Ages 18+

LEGO® MINDSTORMS® NXT
powered by LabVIEW
LabVIEW

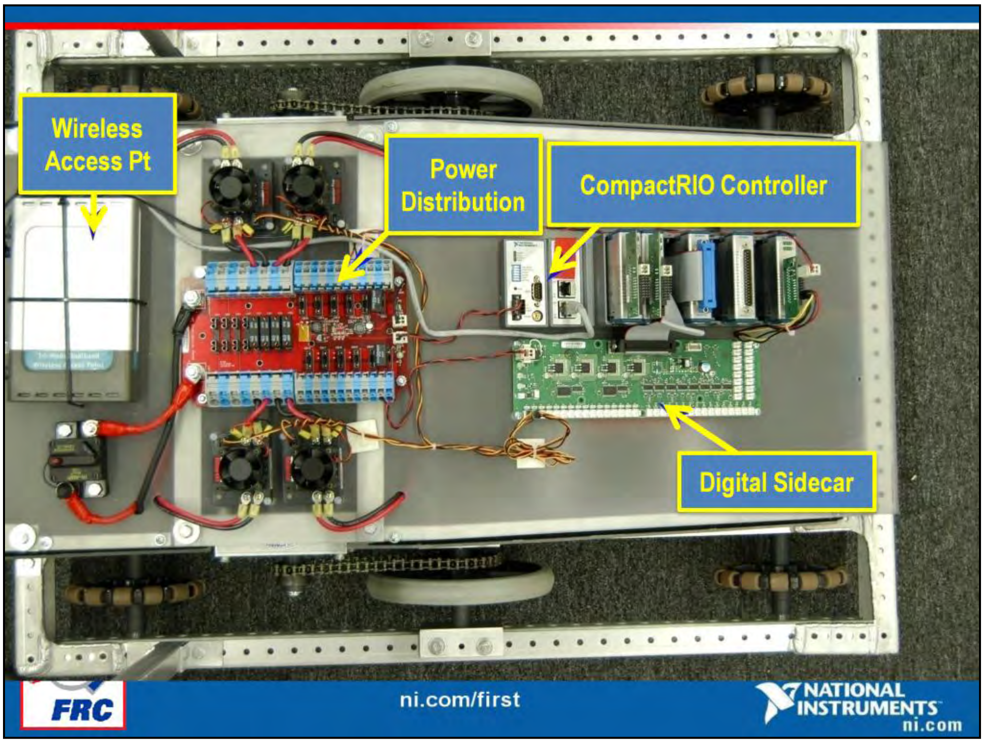
- Students begin to program at age 6 with intuitive, graphical software
- Progressive programming platform—Technology that meets the needs of students of all ages
- Strong industry usage—Strong adoption of LabVIEW and CompactRIO in industry means students develop skills they can use in college and their careers

FRC

ni.com/first

NATIONAL INSTRUMENTS ni.com

With LabVIEW for *FIRST,* students have one programming language they are working with throughout their school careers (from ages 6 to 18)—LabVIEW. If they pursue engineering careers, they can use LabVIEW in more than 5,000 universities across the globe. They will already have a solid foundation in LabVIEW to build on when they begin programming at the university level and into industry.



Wireless Access Pt

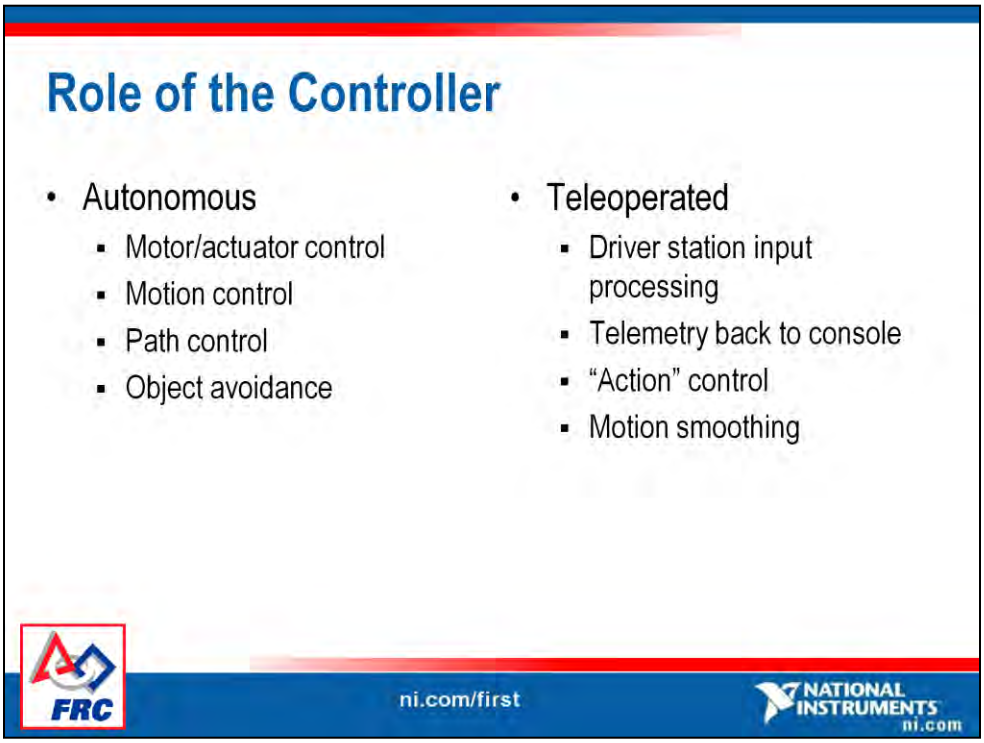
Power Distribution

CompactRIO Controller

Digital Sidecar

ni.com/first

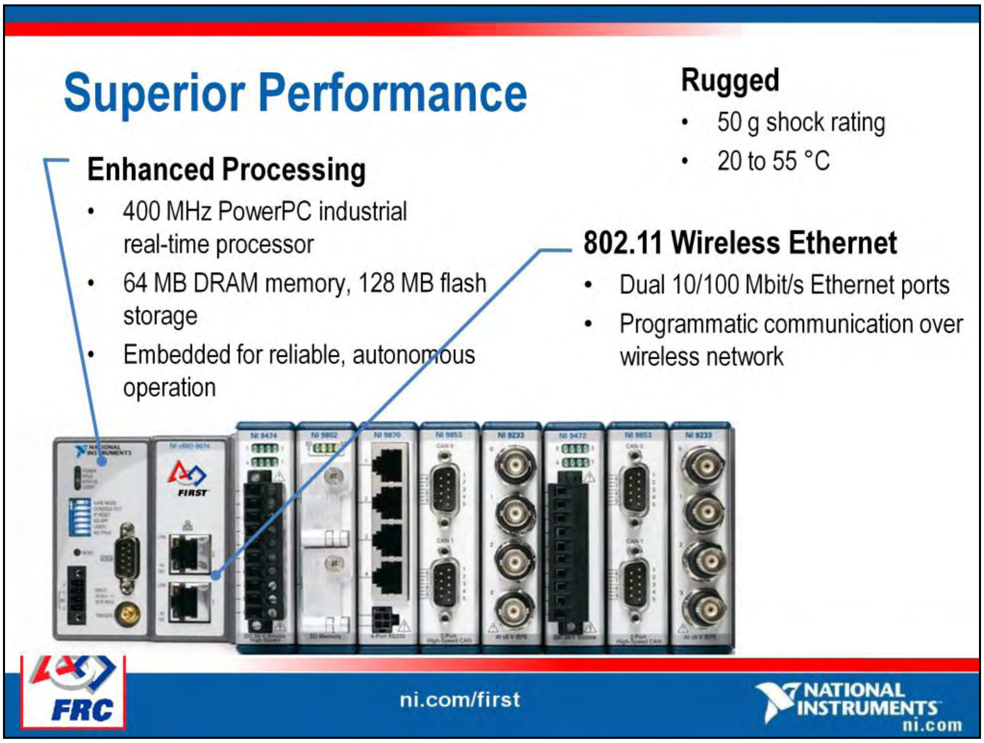
Overall how CompactRIO fits in.



Role of the Controller

- Autonomous
  - Motor/actuator control
  - Motion control
  - Path control
  - Object avoidance
- Teleoperated
  - Driver station input processing
  - Telemetry back to console
  - “Action” control
  - Motion smoothing

ni.com/first



Superior Performance

Enhanced Processing

- 400 MHz PowerPC industrial real-time processor
- 64 MB DRAM memory, 128 MB flash storage
- Embedded for reliable, autonomous operation

Rugged

- 50 g shock rating
- 20 to 55 °C

802.11 Wireless Ethernet

- Dual 10/100 Mbit/s Ethernet ports
- Programmatic communication over wireless network

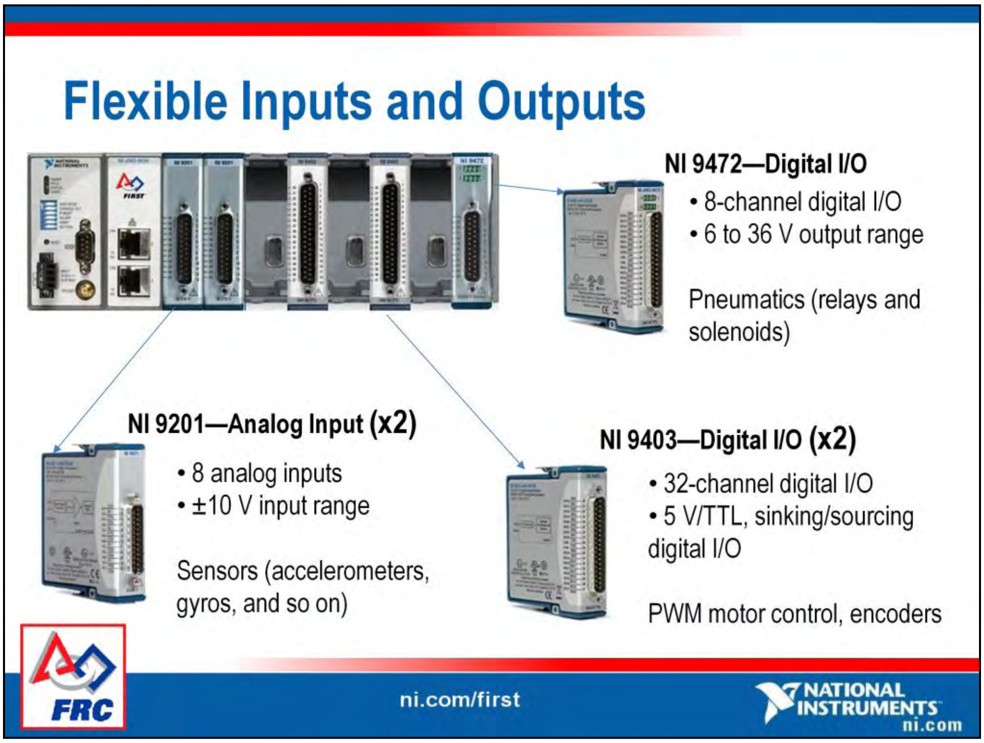
FRC

ni.com/first

NATIONAL INSTRUMENTS
ni.com

Benefits of the *FIRST* CompactRIO Controller

•Students can build sophisticated robotics systems
•Improved autonomous performance
•Advanced vision capabilities
•Multiple sensor choices
•Intuitive, graphical LabVIEW software development will increase participation in autonomous programming
•Industry-standard technology means students gain knowledge for university studies and professional careers



Flexible Inputs and Outputs

NI 9472—Digital I/O
- 8-channel digital I/O
- 6 to 36 V output range

Pneumatics (relays and solenoids)

NI 9201—Analog Input (x2)
- 8 analog inputs
- ±10 V input range

Sensors (accelerometers, gyros, and so on)

NI 9403—Digital I/O (x2)
- 32-channel digital I/O
- 5 V/TTL, sinking/sourcing digital I/O

PWM motor control, encoders

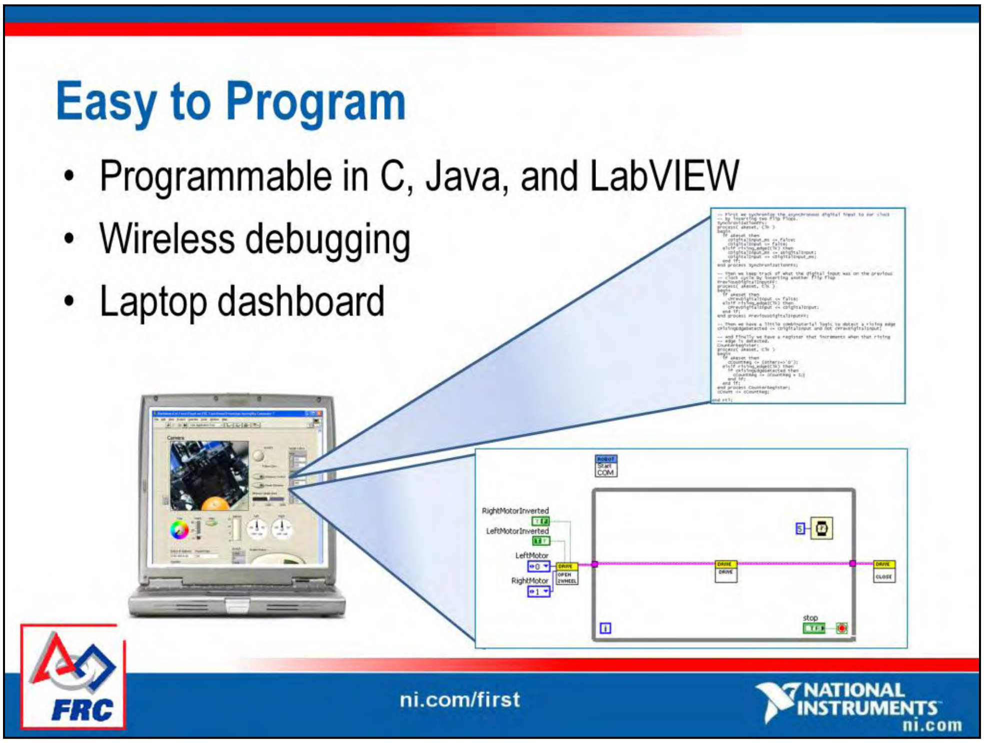
ni.com/first

Note: Modules must be in the same slots in which they shipped for them to work with the FRC LabVIEW code. Please refer to usfirst.org for current slot placement.

NI 9201: For connecting to sensors such as accelerometers, gyros, and so on

NI 9403: For PWM, encoders, and so on

NI 9472: Pneumatic relay for solenoids and relays

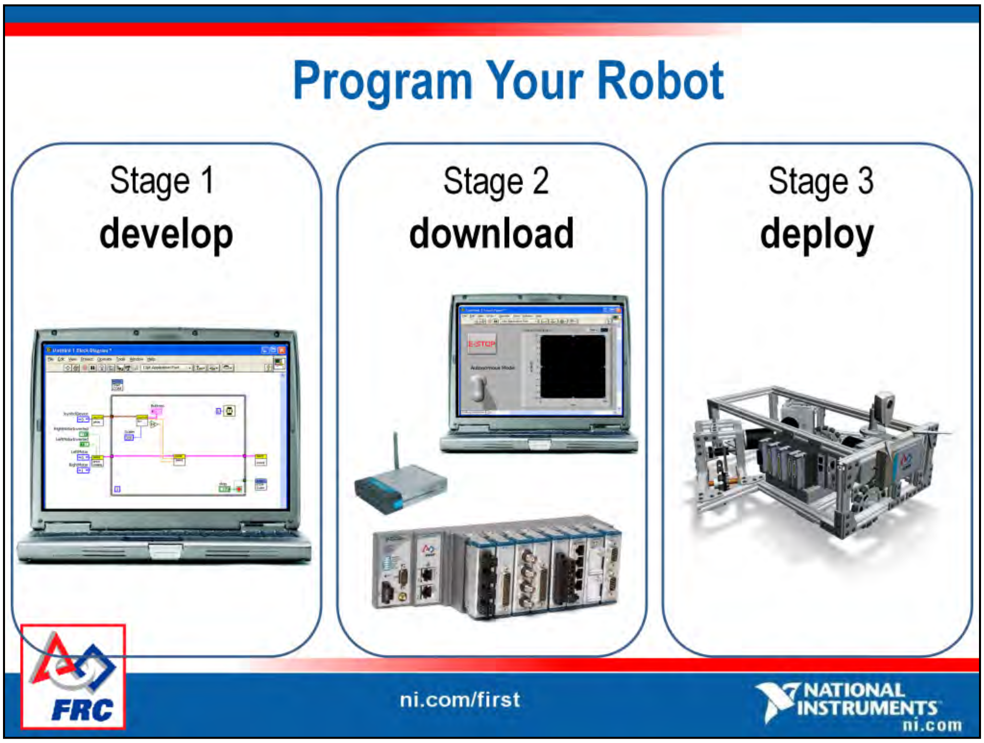


Easy to Program

- Programmable in C, Java, and LabVIEW
- Wireless debugging
- Laptop dashboard

```
-- First we synchronize the asynchronous digital input to our clock
-- by inserting two flip flops.
SynchronizationFFs:
process( aReset, Clk )
begin
  if aReset then
    cDigitalInput_ms <= false;
    cDigitalInput <= false;
  elsif rising_edge(Clk) then
    cDigitalInput_ms <= aDigitalInput;
    cDigitalInput <= cDigitalInput_ms;
  end if;
end process SynchronizationFFs;

-- Then we keep track of what the digital input was on the previous
-- clock cycle by inserting another flip flop
PreviousDigitalInputFF:
process( aReset, Clk )
begin
  if aReset then
    cPrevDigitalInput <= false;
  elsif rising_edge(Clk) then
    cPrevDigitalInput <= cDigitalInput;
  end if;
end process PreviousDigitalInputFF;

-- Then we have a little combinatorial logic to detect a rising edge
cRisingEdgeDetected <= cDigitalInput and not cPrevDigitalInput;

-- And finally we have a register that increments when that rising
-- edge is detected.
CounterRegister:
process( aReset, Clk )
begin
  if aReset then
    cCountReg <= (others=>'0');
  elsif rising_edge(Clk) then
    if cRisingEdgeDetected then
      cCountReg <= cCountReg + 1;
    end if;
  end if;
end process CounterRegister;
cCount <= cCountReg;

end rtl;
```

ni.com/first



Program Your Robot

Stage 1
develop

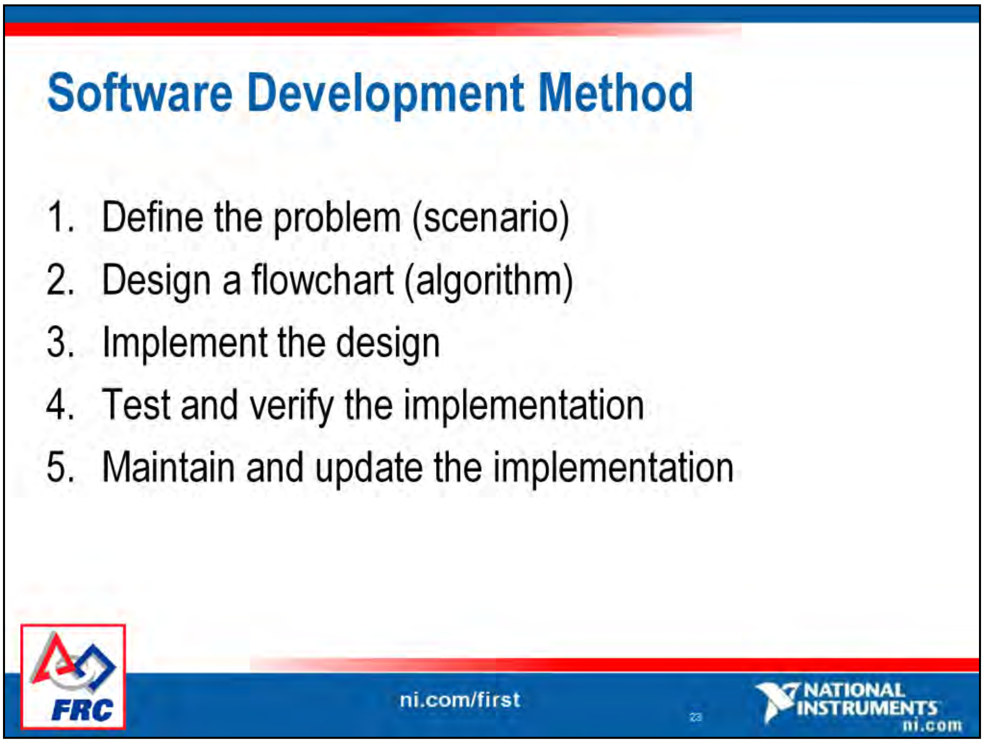
Stage 2
download

Stage 3
deploy

Software Development Fundamentals

ni.com/first

Software Development Fundamentals: Coaching students to implement flowcharts or state transition diagrams to meet control application challenges



Software Development Method

1. Define the problem (scenario)
2. Design a flowchart (algorithm)
3. Implement the design
4. Test and verify the implementation
5. Maintain and update the implementation

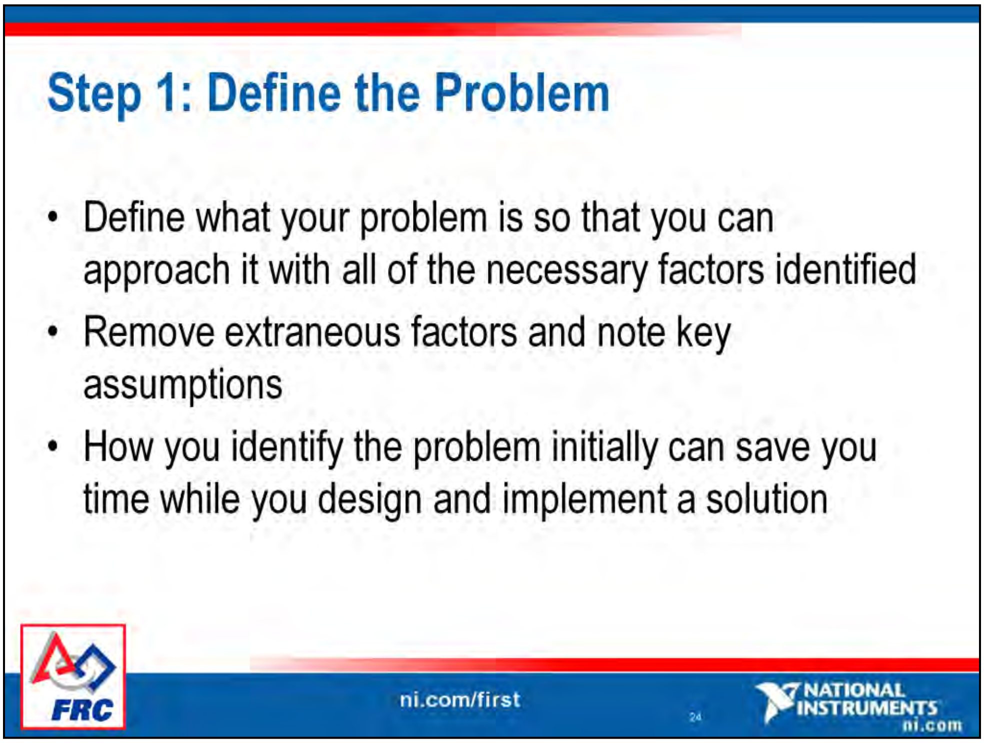
ni.com/first

Following a set of steps that has been refined over the years by software engineers can simplify solving problems using software. The software development method is a strategy for using LabVIEW to implement a software solution. Use the software development method to create a solution to your problem.

In the software development method, complete the following steps:

1. Define the problem (scenario)
2. Design a flowchart (algorithm)
3. Implement the design
4. Test and verify the implementation
5. Maintain and update the implementation

For more information on this process, please visit http://zone.ni.com/devzone/cda/tut/p/id/10051.



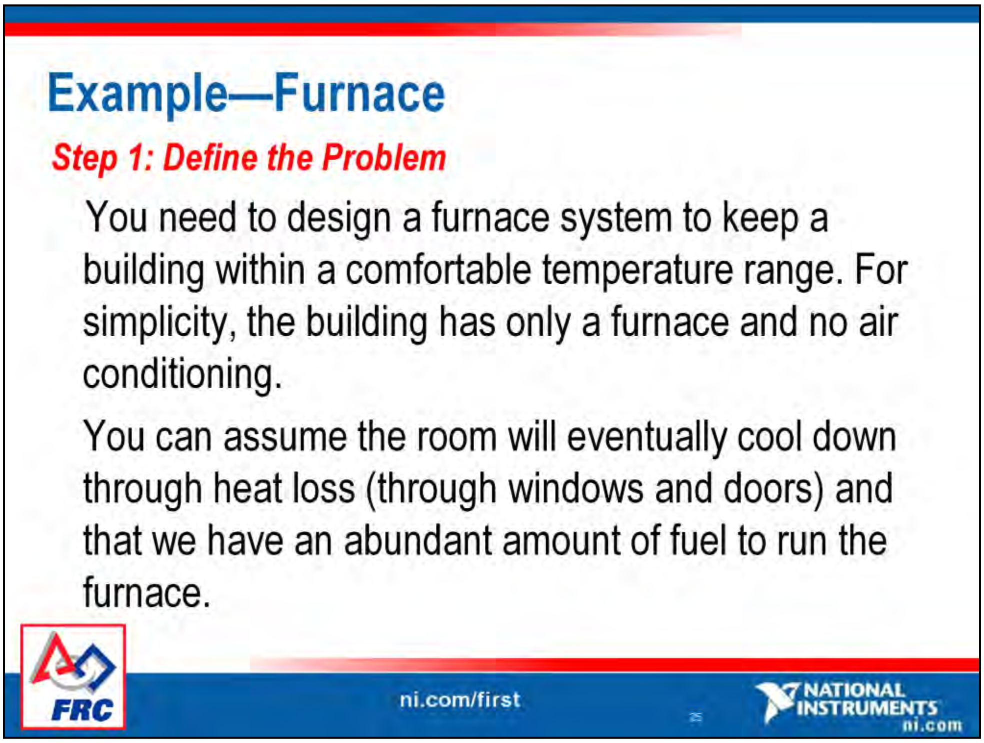
Step 1: Define the Problem

- Define what your problem is so that you can approach it with all of the necessary factors identified
- Remove extraneous factors and note key assumptions
- How you identify the problem initially can save you time while you design and implement a solution

ni.com/first

24

During the problem (scenario) stage of the software development method, you define what your problem is so that you can approach it with all of the necessary factors identified. You can remove extraneous factors during this phase and focus on the core problem that you must solve. How you identify the problem initially can save you time while you design and implement a solution.



Example—Furnace

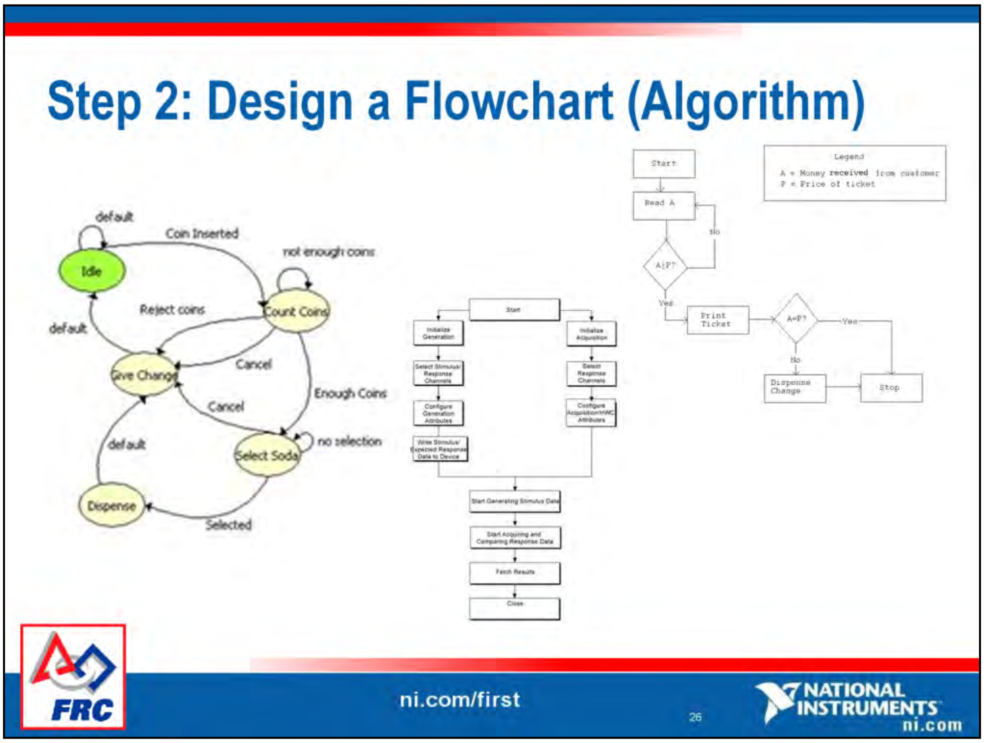
Step 1: Define the Problem

You need to design a furnace system to keep a building within a comfortable temperature range. For simplicity, the building has only a furnace and no air conditioning.

You can assume the room will eventually cool down through heat loss (through windows and doors) and that we have an abundant amount of fuel to run the furnace.

ni.com/first

During the problem (scenario) stage of the software development method, you define what your problem is so that you can approach it with all of the necessary factors identified. You can remove extraneous factors during this phase and focus on the core problem that you must solve. How you identify the problem initially can save you time while you design and implement a solution.



Step 2: Design a Flowchart (Algorithm)

ni.com/first

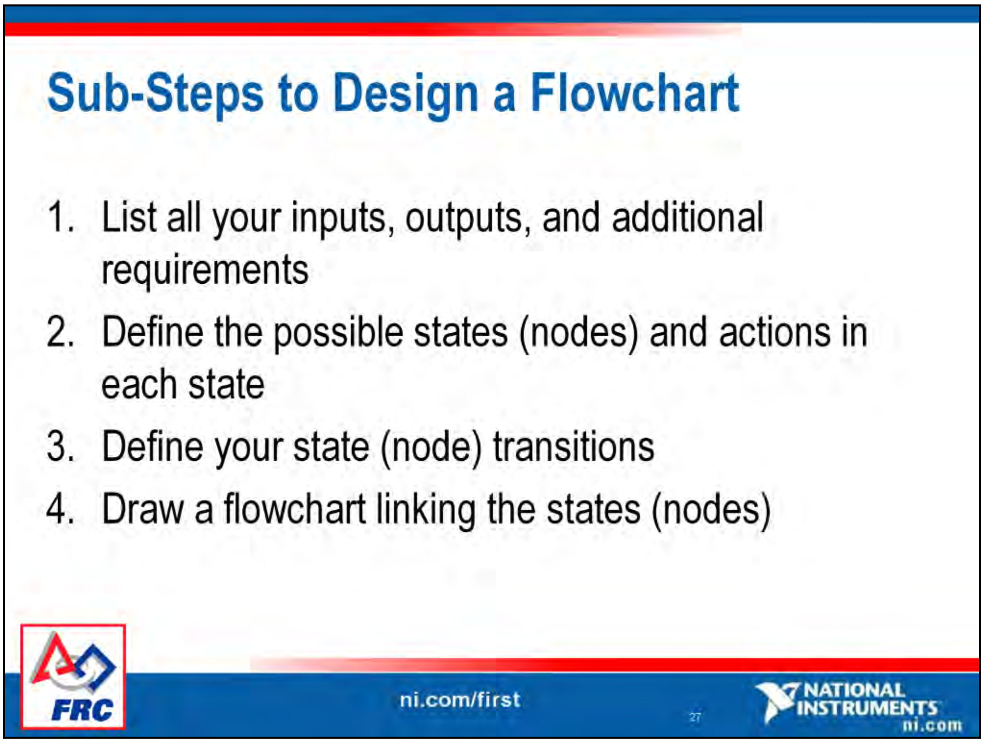
26

NATIONAL INSTRUMENTS ni.com

The first place to start with any software program is to map out a flowchart of how the process will execute. The "process" could be a robot, a control loop, or basically any operation that has a series of steps that need to be completed.

The image above shows a number of different flowcharts. The format/type of the flowchart is less important than the act of thinking through and drawing out the flowchart.

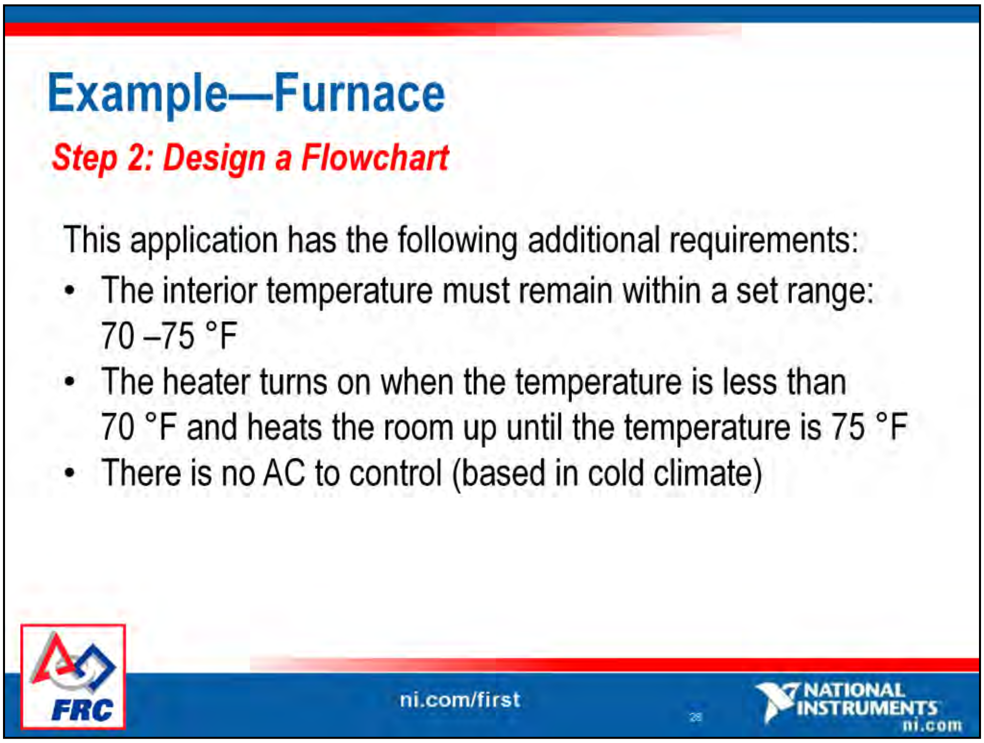
Flowcharts help you develop a good understanding of the application flow by dividing the application into more manageable pieces called **states** or **nodes**. NI LabVIEW is a graphical programming language designed for measurement and automation applications, which makes it an ideal tool for quickly converting your paper design to code. Since a LabVIEW block diagram is similar to a flowchart, moving from the flowchart to software code is a quick process.



Sub-Steps to Design a Flowchart

1. List all your inputs, outputs, and additional requirements
2. Define the possible states (nodes) and actions in each state
3. Define your state (node) transitions
4. Draw a flowchart linking the states (nodes)

FRC
ni.com/first
27
NATIONAL INSTRUMENTS
ni.com



Example—Furnace

Step 2: Design a Flowchart

This application has the following additional requirements:

- The interior temperature must remain within a set range: 70 –75 °F
- The heater turns on when the temperature is less than 70 °F and heats the room up until the temperature is 75 °F
- There is no AC to control (based in cold climate)

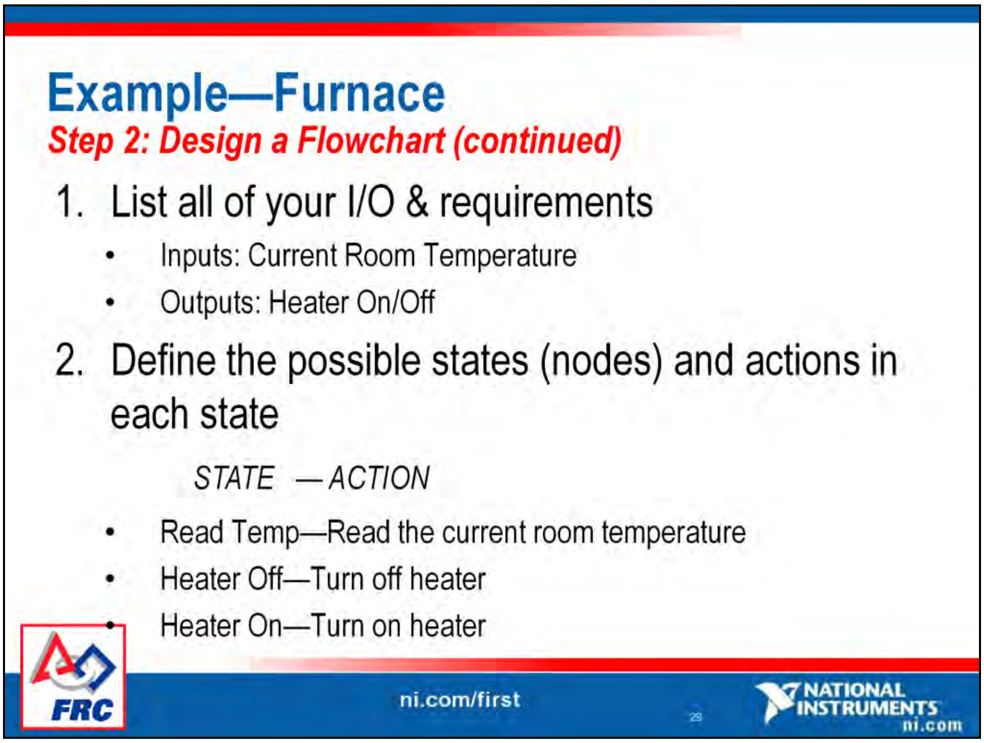
ni.com/first

28

Remember…

Sub-Steps to Design a Flowchart

1. List all your inputs, outputs, and additional requirements
2. Define the possible states (nodes) and actions in each state
3. Define your state (node) transitions
4. Draw a flowchart linking the states (nodes)



Example—Furnace

Step 2: Design a Flowchart (continued)

1. List all of your I/O & requirements
   - Inputs: Current Room Temperature
   - Outputs: Heater On/Off
2. Define the possible states (nodes) and actions in each state

   STATE —ACTION

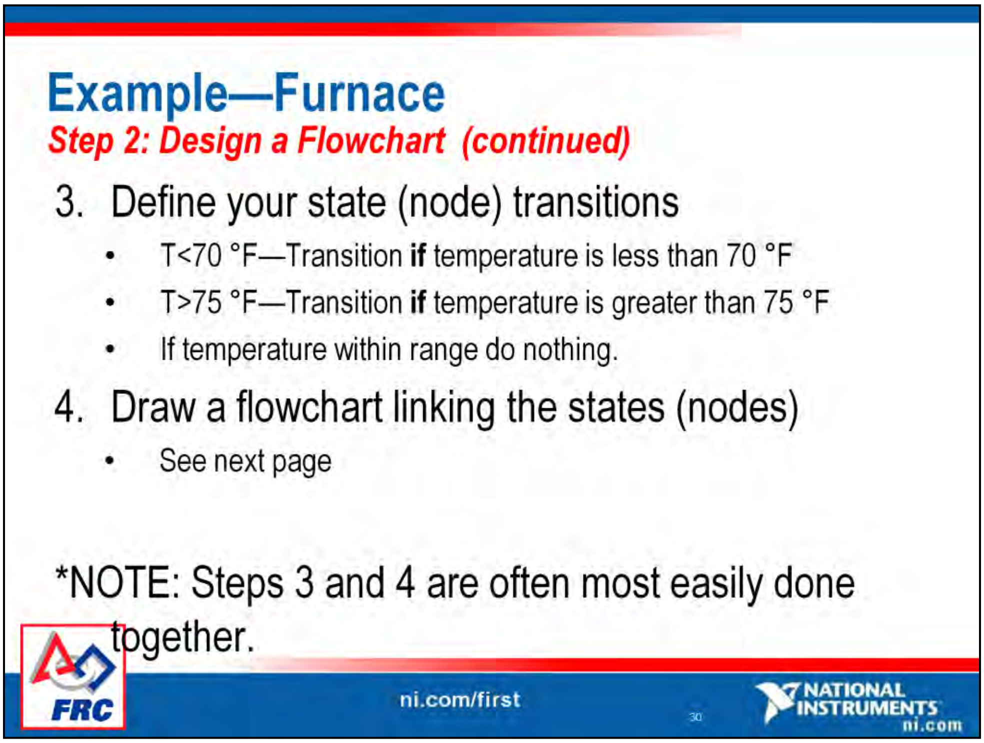
   - Read Temp—Read the current room temperature
   - Heater Off—Turn off heater
   - Heater On—Turn on heater

FRC
ni.com/first
29
NATIONAL INSTRUMENTS
ni.com

1. List all of your inputs, outputs, and additional requirements
   - Inputs: Current Room Temperature
     - Since the temperature range is fixed at 70-75 deg F, the furnace set point temperature is not considered.
     - Assuming furnace is powered on and fuel (electricity or gas) is provided.
   - Outputs: Heater On/Off
   - Requirements: See above.

2. Define the possible states (nodes) and actions in each state
   - Read Temp—Read the current room temperature
   - Heater Off—Turn off heater
   - Heater On—Turn on heater

   The more states you consider and add, generally the more robust the program/process will be. In many cases, additional states are added later during the code debug as scenarios not considered are encountered. For example, what happens when we need to shut down the furnace before the temperature reaches 75 °F for maintenance or if it is summer time? Here we could add the following new input, state, and action: Emergency Stop Button (input), E-Button Pressed (state), and Turn Off Heater and Finish (action).



Example—Furnace

Step 2: Design a Flowchart (continued)

3. Define your state (node) transitions
   - T<70 °F—Transition **if** temperature is less than 70 °F
   - T>75 °F—Transition **if** temperature is greater than 75 °F
   - If temperature within range do nothing.
4. Draw a flowchart linking the states (nodes)
   - See next page

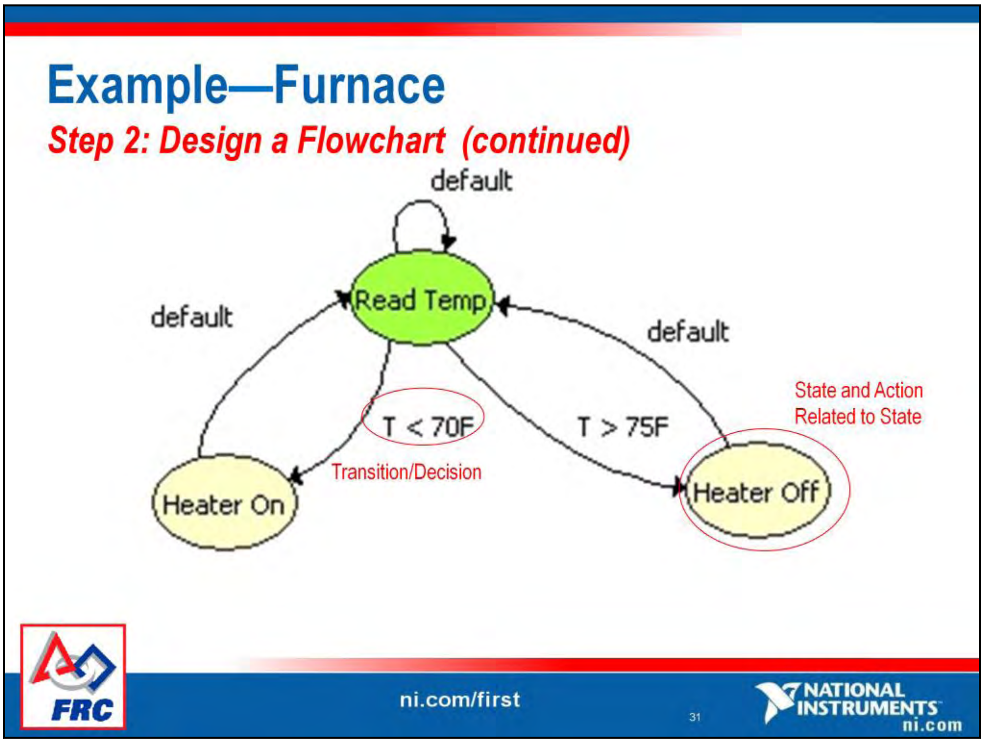
*NOTE: Steps 3 and 4 are often most easily done together.

ni.com/first

3. Define your state (node) transitions
   - T<70 °F—Transition if temperature is less than 70 °F
   - T>75 °F—Transition if temperature is greater than 75 °F
   - If temperature within range do nothing

4. Draw a flowchart linking the states (nodes)

*NOTE: Steps 3 and 4 are often most easily done together.



Example—Furnace

Step 2: Design a Flowchart (continued)

State and Action Related to State

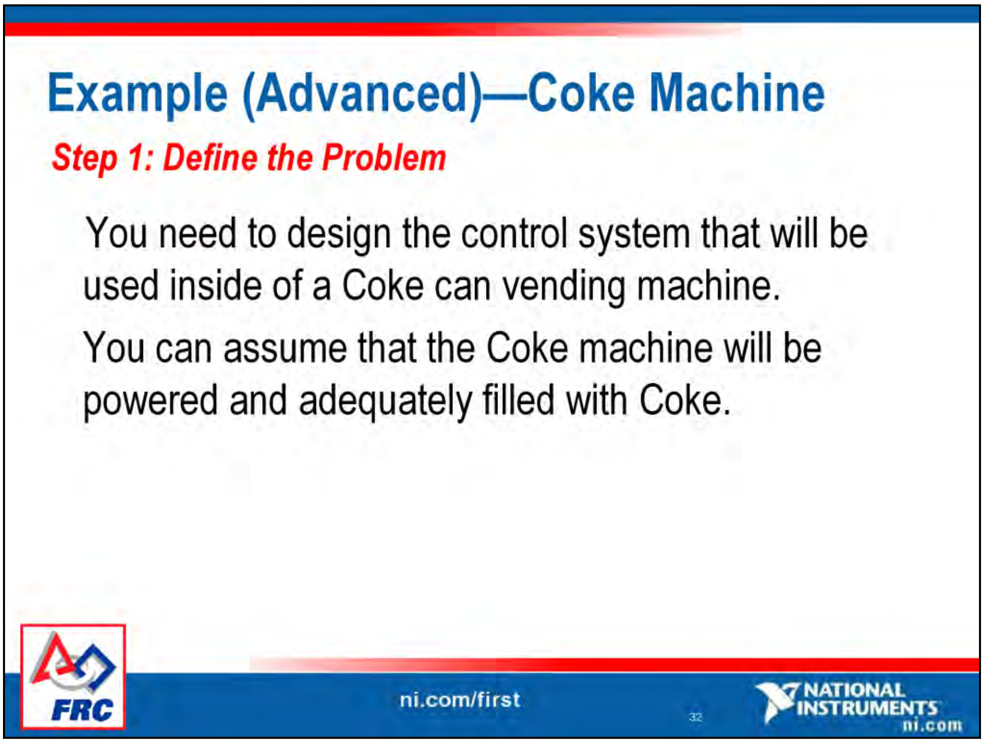
Transition/Decision

ni.com/first

31

The more states you consider and add, generally the more robust the program/process will be. In many cases, additional states are added later during the code debug as scenarios not considered are encountered.

For example, what happens when we need to shut down the furnace before the temperature reaches 75 °F for maintenance or if it is summer time? Here we could add the following new input, state, and action: Emergency Stop Button (input), E-Button Pressed (state), and Turn Off Heater and Finish (action).



Example (Advanced)—Coke Machine

Step 1: Define the Problem

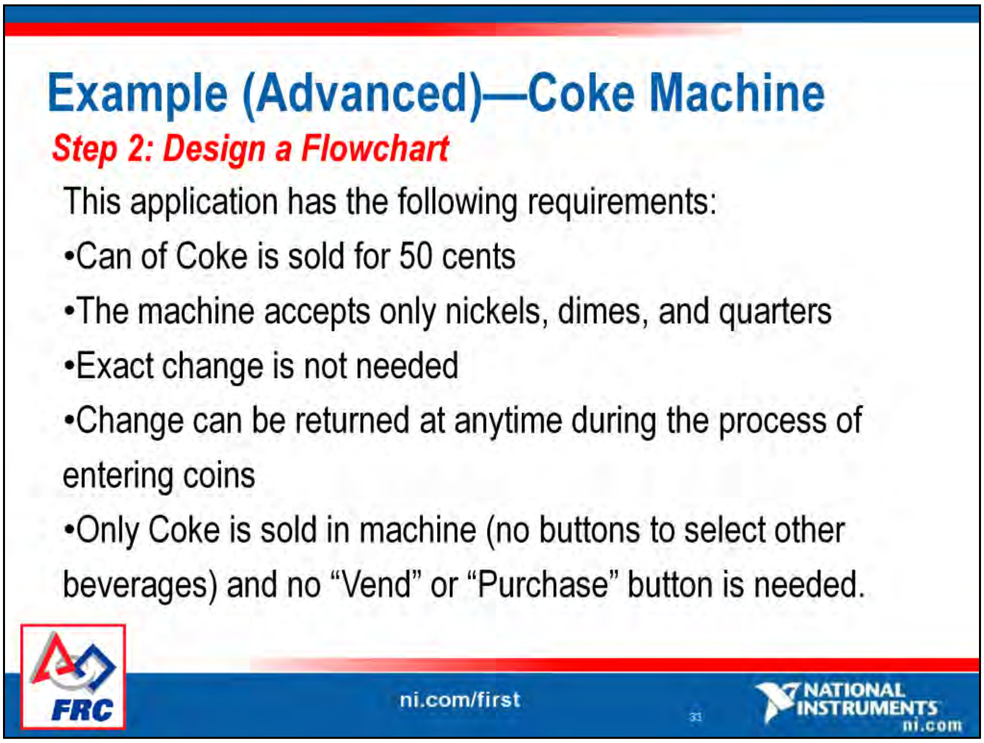
You need to design the control system that will be used inside of a Coke can vending machine.

You can assume that the Coke machine will be powered and adequately filled with Coke.

ni.com/first

32

During the problem (scenario) stage of the software development method, you define what your problem is so that you can approach it with all of the necessary factors identified. You can remove extraneous factors during this phase and focus on the core problem that you must solve. How you identify the problem initially can save you time while you design and implement a solution.



Example (Advanced)—Coke Machine

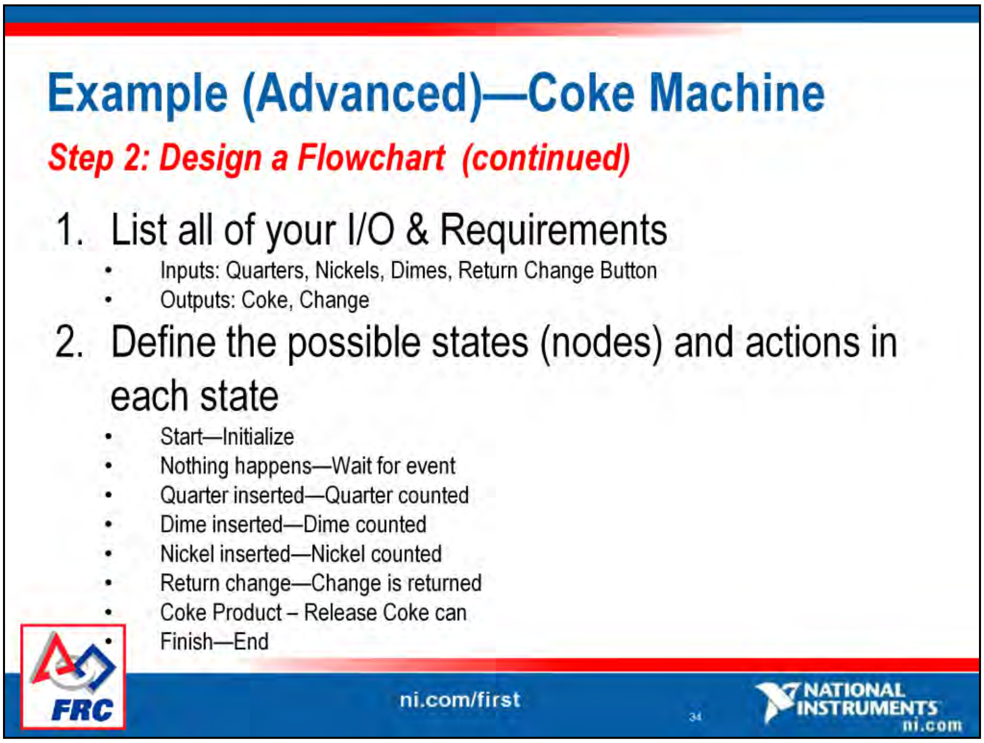
Step 2: Design a Flowchart

This application has the following requirements:

- Can of Coke is sold for 50 cents
- The machine accepts only nickels, dimes, and quarters
- Exact change is not needed
- Change can be returned at anytime during the process of entering coins
- Only Coke is sold in machine (no buttons to select other beverages) and no “Vend” or “Purchase” button is needed.

ni.com/first

33



Example (Advanced)—Coke Machine

Step 2: Design a Flowchart (continued)

1. List all of your I/O & Requirements
   - Inputs: Quarters, Nickels, Dimes, Return Change Button
   - Outputs: Coke, Change
2. Define the possible states (nodes) and actions in each state
   - Start—Initialize
   - Nothing happens—Wait for event
   - Quarter inserted—Quarter counted
   - Dime inserted—Dime counted
   - Nickel inserted—Nickel counted
   - Return change—Change is returned
   - Coke Product – Release Coke can
   - Finish—End

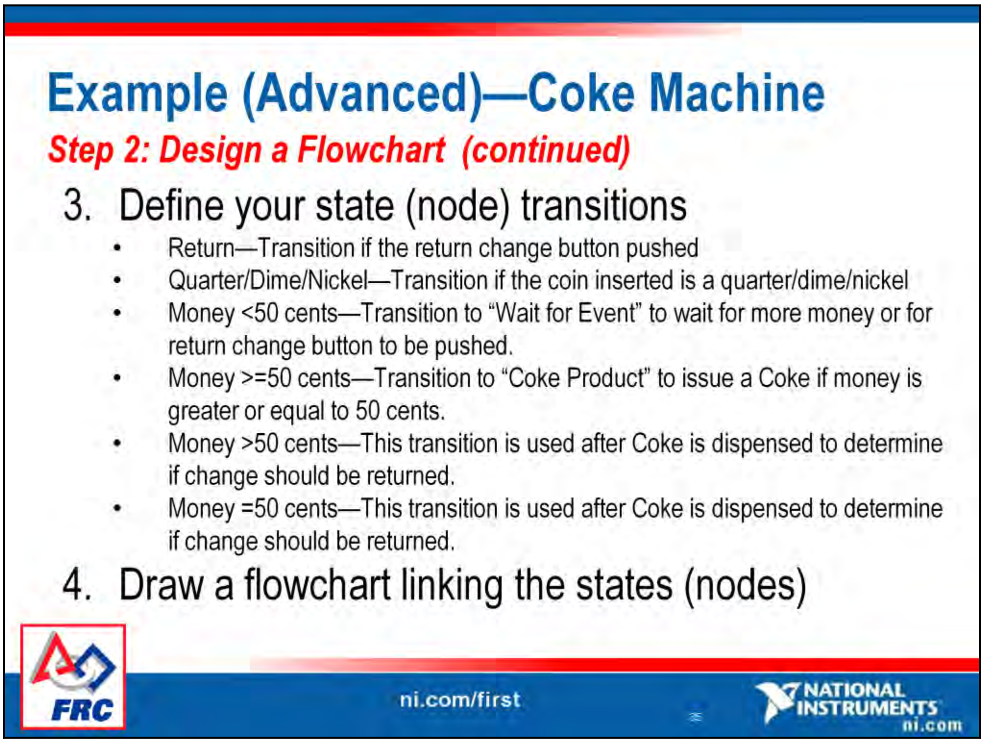
ni.com/first

34

1. List all of your inputs, outputs, and additional requirements
   - Inputs: Quarters, Nickels, Dimes, Return Change Button
   - Outputs: Coke, Change
   - Requirements: See above.

2. Define the possible states (nodes) and actions in each state
   - Start—Initialize
   - Nothing happens—Wait for event
   - Quarter inserted—Quarter counted
   - Dime inserted—Dime counted
   - Nickel inserted—Nickel counted
   - Return change—Change is returned
   - Coke Product — Release Coke can
   - Finish—End

The more states considered and added, generally the more robust the program/process will be. In many cases, additional states are added later during the code debug as scenarios not considered are encountered. For example, what happens if a damaged coin or fake coin is inserted? The scenario (state) is not considered in this example, but could be included with another state “Unknown coin or object – return coin or object”. Try to explore other unconsidered scenarios with the students. Another example, what happens if the machine is out of Coke!?



Example (Advanced)—Coke Machine

Step 2: Design a Flowchart (continued)

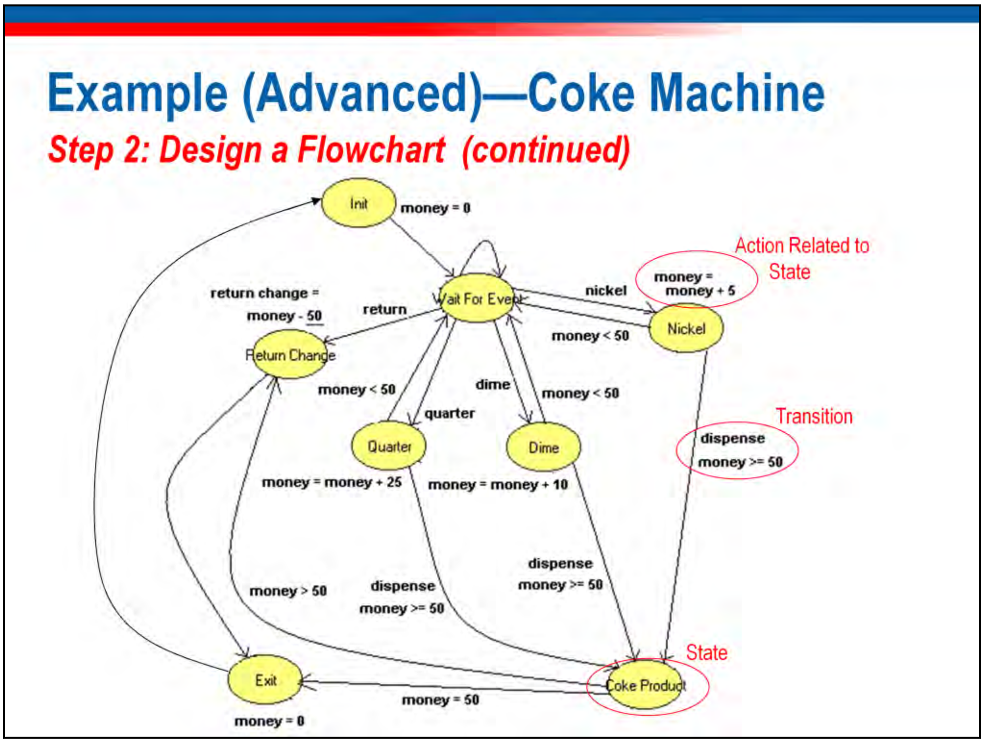
3. Define your state (node) transitions
   - Return—Transition if the return change button pushed
   - Quarter/Dime/Nickel—Transition if the coin inserted is a quarter/dime/nickel
   - Money <50 cents—Transition to “Wait for Event” to wait for more money or for return change button to be pushed.
   - Money >=50 cents—Transition to “Coke Product” to issue a Coke if money is greater or equal to 50 cents.
   - Money >50 cents—This transition is used after Coke is dispensed to determine if change should be returned.
   - Money =50 cents—This transition is used after Coke is dispensed to determine if change should be returned.
4. Draw a flowchart linking the states (nodes)

ni.com/first

3. Define your state (node) transitions
   - Return—Transition if the return change button pushed
   - Quarter/Dime/Nickel—Transition if the coin inserted is a quarter/dime/nickel
   - Money <50 cents—Transition to “Wait for Event” to wait for more money or for return change button to be pushed.
   - Money >=50 cents—Transition to “Coke Product” to issue a Coke if money is greater or equal to 50 cents.
   - Money >50 cents—This transition is used after Coke is dispensed to determine if change should be returned.
   - Money =50 cents—This transition is used after Coke is dispensed to determine if change should be returned.

4. Draw a flowchart linking the states (nodes)

*NOTE: Steps 3 and 4 are often most easily done together.



Example (Advanced)—Coke Machine

Step 2: Design a Flowchart (continued)

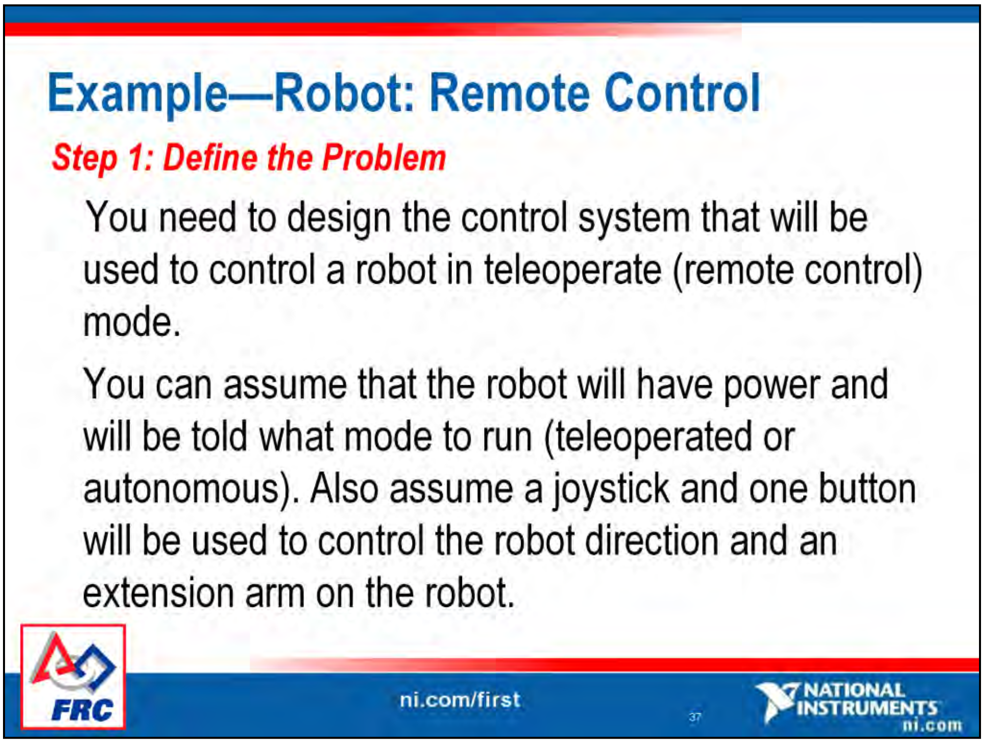
Action Related to State

Transition

State

More detail of this example. http://zone.ni.com/devzone/cda/tut/p/id/3024

The more states considered and added, generally the more robust the program/process will be. In many cases, additional states are added later during the code debug as scenarios not considered are encountered. For example, what happens if a damaged coin or fake coin is inserted? The scenario (state) is not considered in this example, but could be included with another state “Unknown coin or object – return coin or object”. Try to explore other unconsidered scenarios with the students. Another example, what happens if the machine is out of Coke!?



Example—Robot: Remote Control

Step 1: Define the Problem

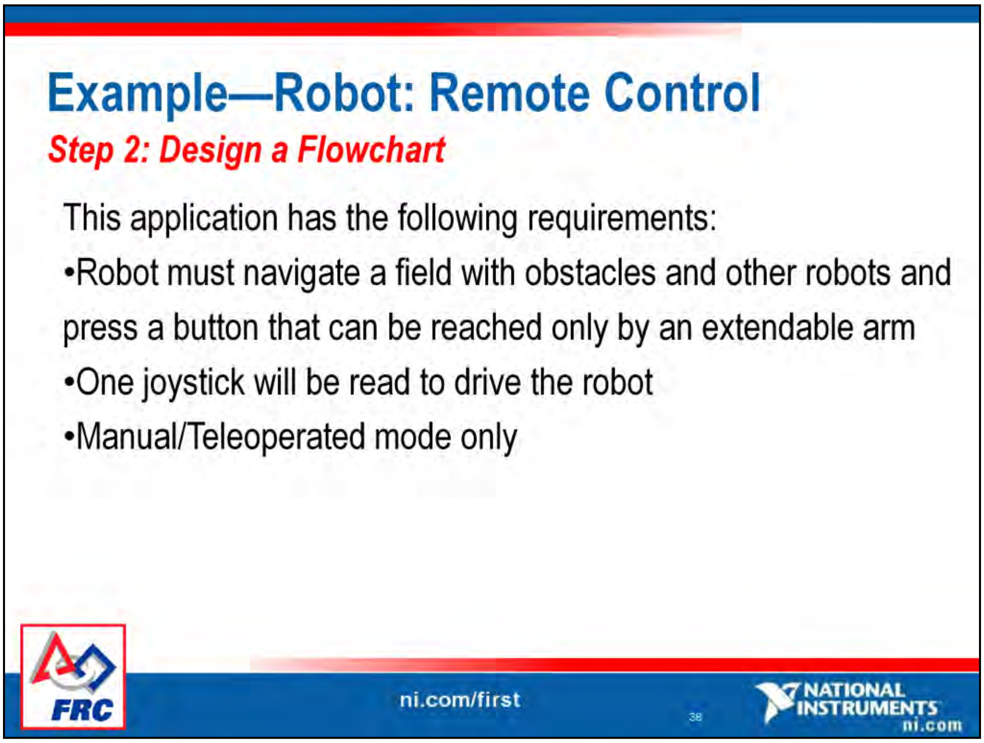
You need to design the control system that will be used to control a robot in teleoperate (remote control) mode.

You can assume that the robot will have power and will be told what mode to run (teleoperated or autonomous). Also assume a joystick and one button will be used to control the robot direction and an extension arm on the robot.

ni.com/first

37

During the problem (scenario) stage of the software development method, you define what your problem is so that you can approach it with all of the necessary factors identified. You can remove extraneous factors during this phase and focus on the core problem that you must solve. How you identify the problem initially can save you time while you design and implement a solution.



Example—Robot: Remote Control

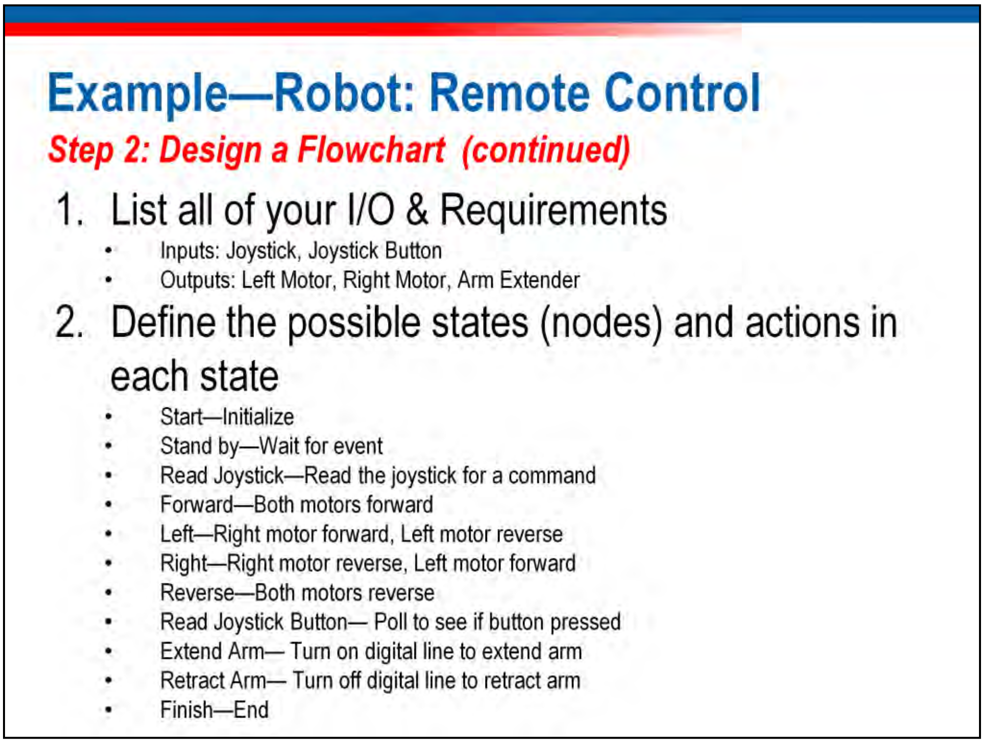
Step 2: Design a Flowchart

This application has the following requirements:

- Robot must navigate a field with obstacles and other robots and press a button that can be reached only by an extendable arm
- One joystick will be read to drive the robot
- Manual/Teleoperated mode only

ni.com/first

38



Example—Robot: Remote Control

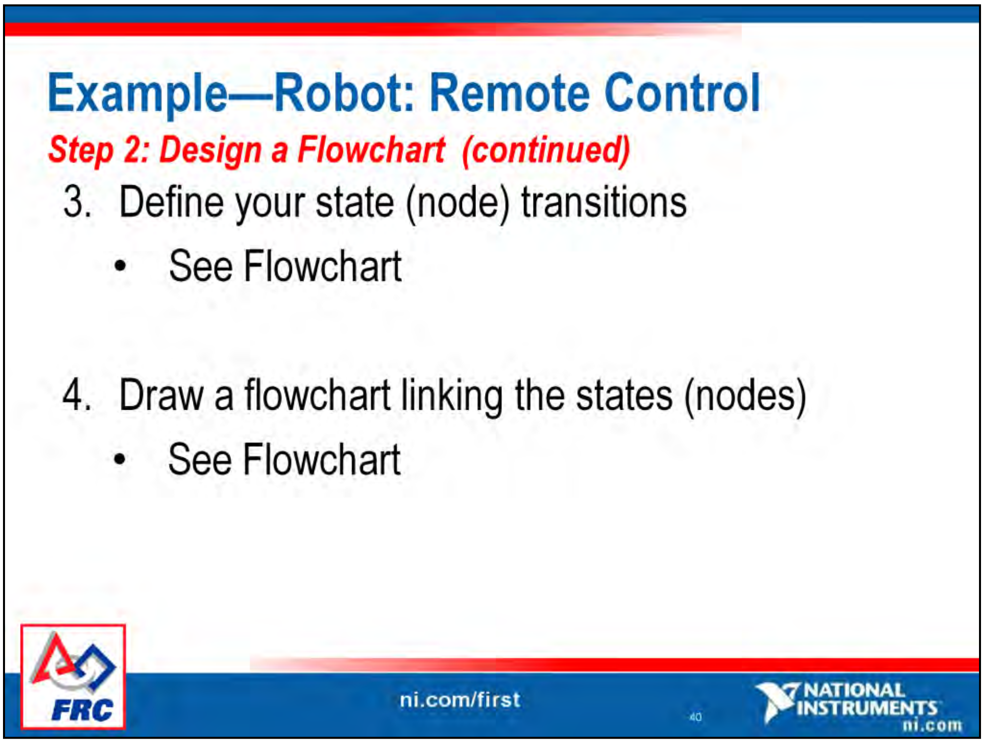
Step 2: Design a Flowchart (continued)

1. List all of your I/O & Requirements
   - Inputs: Joystick, Joystick Button
   - Outputs: Left Motor, Right Motor, Arm Extender
2. Define the possible states (nodes) and actions in each state
   - Start—Initialize
   - Stand by—Wait for event
   - Read Joystick—Read the joystick for a command
   - Forward—Both motors forward
   - Left—Right motor forward, Left motor reverse
   - Right—Right motor reverse, Left motor forward
   - Reverse—Both motors reverse
   - Read Joystick Button— Poll to see if button pressed
   - Extend Arm— Turn on digital line to extend arm
   - Retract Arm— Turn off digital line to retract arm
   - Finish—End

MANUAL MODE – Teleoperated

1. List all of your inputs, outputs, and additional requirements
   - Inputs: Joystick, Joystick Button
   - Outputs: Left Motor, Right Motor, Arm Extender
   - Requirements: See above.

2. Define the possible states (nodes) and actions in each state
   - Start—Initialize
   - Stand by—Wait for event
   - Read Joystick—Read the joystick for a command
   - Forward—Both motors forward
   - Left—Right motor forward, Left motor reverse
   - Right—Right motor reverse, Left motor forward
   - Reverse—Both motors reverse
   - Read Joystick Button— Poll to see if button pressed
   - Extend Arm— Turn on digital line to extend arm
   - Retract Arm— Turn off digital line to retract arm
   - Finish—End



Example—Robot: Remote Control

Step 2: Design a Flowchart (continued)

3. Define your state (node) transitions
   - See Flowchart
4. Draw a flowchart linking the states (nodes)
   - See Flowchart

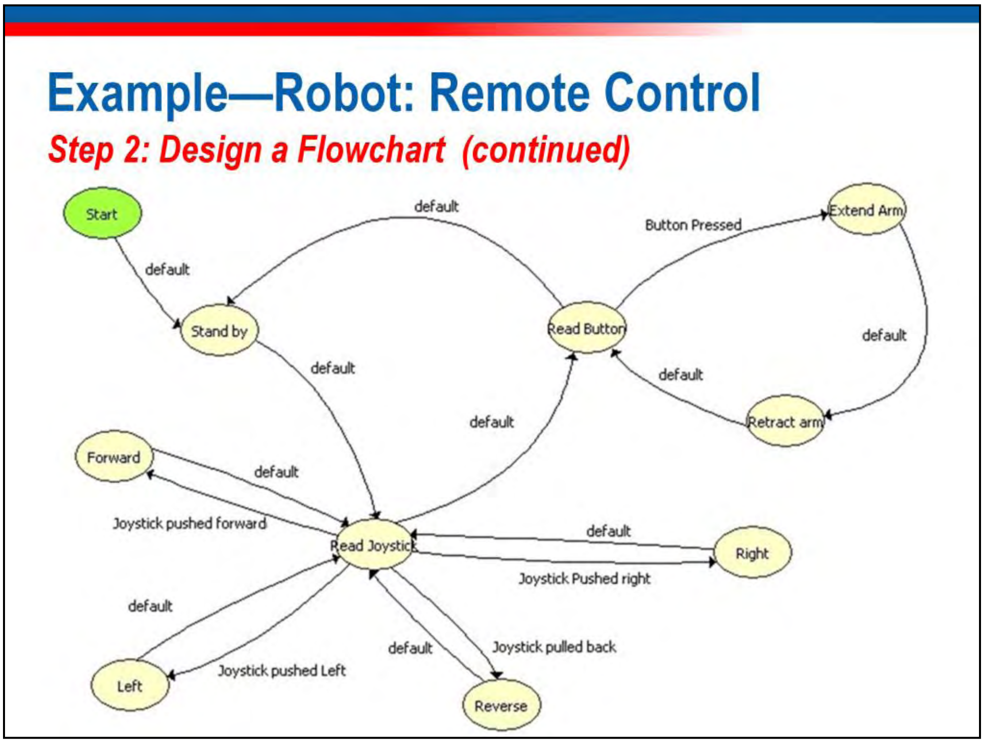
FRC

ni.com/first

40

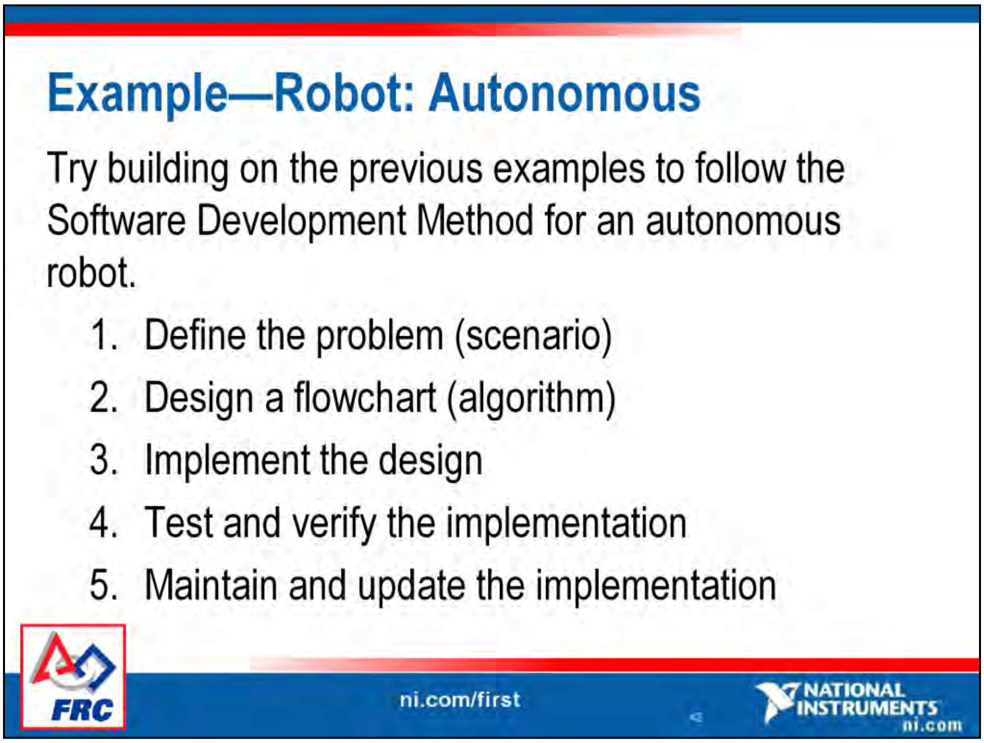
NATIONAL INSTRUMENTS ni.com

*NOTE: Steps 3 and 4 are often most easily done together.



Example—Robot: Remote Control

Step 2: Design a Flowchart (continued)



Example—Robot: Autonomous

Try building on the previous examples to follow the Software Development Method for an autonomous robot.

1. Define the problem (scenario)
2. Design a flowchart (algorithm)
3. Implement the design
4. Test and verify the implementation
5. Maintain and update the implementation

FRC

ni.com/first

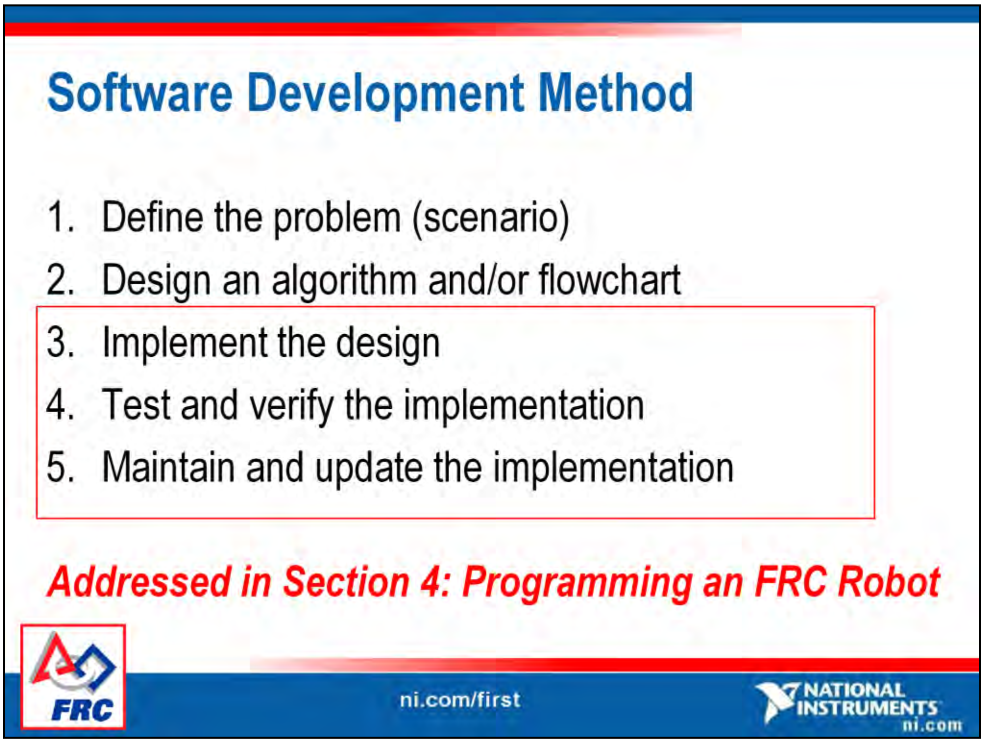
NATIONAL INSTRUMENTS ni.com

AUTONOMOUS MODE – Step 2: Design a flowchart sub-steps shown below.

1. List all of your inputs, outputs, and additional requirements
   - Inputs: Line Follower, Linear Encoder, Rotary Encoder for Each Drive Motor, Proximity Sensor
   - Outputs: Left Motor, Right Motor, Arm Extender
   - This application has the following requirements:
     - ▪Robot must navigate a field with obstacles and other robots and press a button that can be reached only by an extendable arm
     - ▪ There is a line on the course that marks a clear path from the robot’s starting position to the button’s location
     - ▪Autonomous mode only
2. Define the possible states (nodes) and actions in each state
   - Start—Initialize
   - Read Line—Get reading from line sensor to get position on line
   - On Line—Left and right drive motors forward at same speed
   - To Left of Line—Right motor 75% speed of left motor
   - To Right of Line—Left motor 75% speed of right motor
   - Run into Wall—Extend arm
   - Arm Extended to Required Distance—Retract arm
   - Finish—End

   The more states you consider and add, generally the more robust the program/process will be. In many cases, additional states are added later during the code debug as scenarios not considered are encountered. For example, what happens if another robot is in your robot’s path? The scenario (state) is not considered in this example, but it could be included with another state “Robot Obstruction.” Try to explore other unconsidered scenarios with the students. Another example is what happens if the robot cannot find the line in the first place?
3. Define your state (node) transitions
4. Draw a flowchart linking the states (nodes)

*NOTE: Steps 3 and 4 are often most easily done together.



Software Development Method

1. Define the problem (scenario)
2. Design an algorithm and/or flowchart
3. Implement the design
4. Test and verify the implementation
5. Maintain and update the implementation

Addressed in Section 4: Programming an FRC Robot

FRC

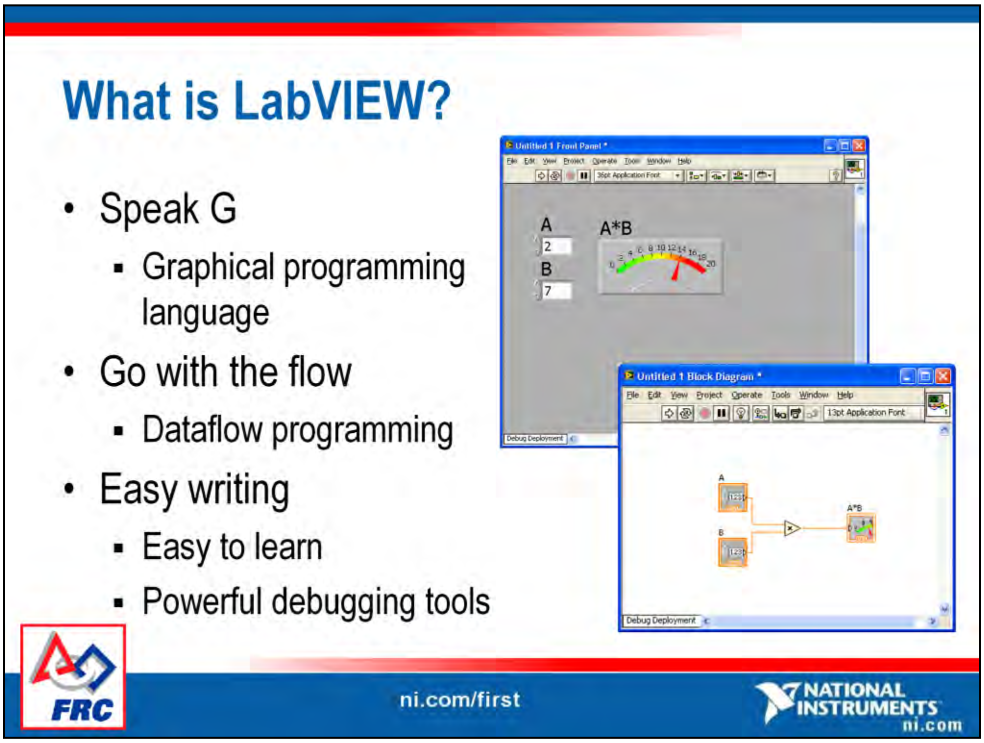
ni.com/first

NATIONAL INSTRUMENTS
ni.com

Introduction to NI LabVIEW

ni.com/first

Introduction to NI LabVIEW: Overview of the LabVIEW environment, dataflow programming, and common programming structures used in the FRC code template



What is LabVIEW?

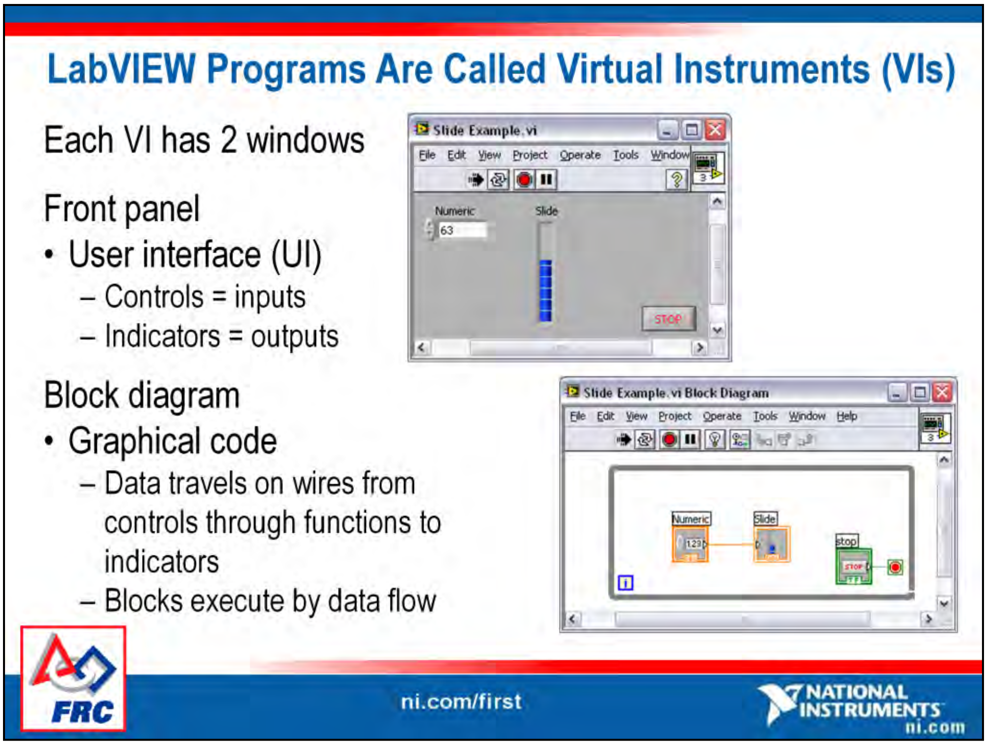
- Speak G
  - Graphical programming language
- Go with the flow
  - Dataflow programming
- Easy writing
  - Easy to learn
  - Powerful debugging tools

ni.com/first

LabVIEW is a graphical programming language—you draw your program. It is easy to learn and use while being a powerful, full-featured programming language. LabVIEW uses data flow, meaning that a node does not run until it has all of the data from previous nodes at its input wires.

A LabVIEW program consists of one or more virtual instruments (VIs).

A VI is like a function or a procedure or a subroutine—except that it has a richer interface.



LabVIEW Programs Are Called Virtual Instruments (VIs)

Each VI has 2 windows

Front panel
- User interface (UI)
  - Controls = inputs
  - Indicators = outputs

Block diagram
- Graphical code
  - Data travels on wires from controls through functions to indicators
  - Blocks execute by data flow

Slide Example.vi
File Edit View Project Operate Tools Window
Numeric
63
Slide
STOP
Slide Example.vi Block Diagram
File Edit View Project Operate Tools Window Help
Numeric
123
Slide
stop
STOP
FRC
ni.com/first
NATIONAL INSTRUMENTS
ni.com

LabVIEW programs are called virtual instruments (VIs).

Controls are inputs and indicators are outputs.

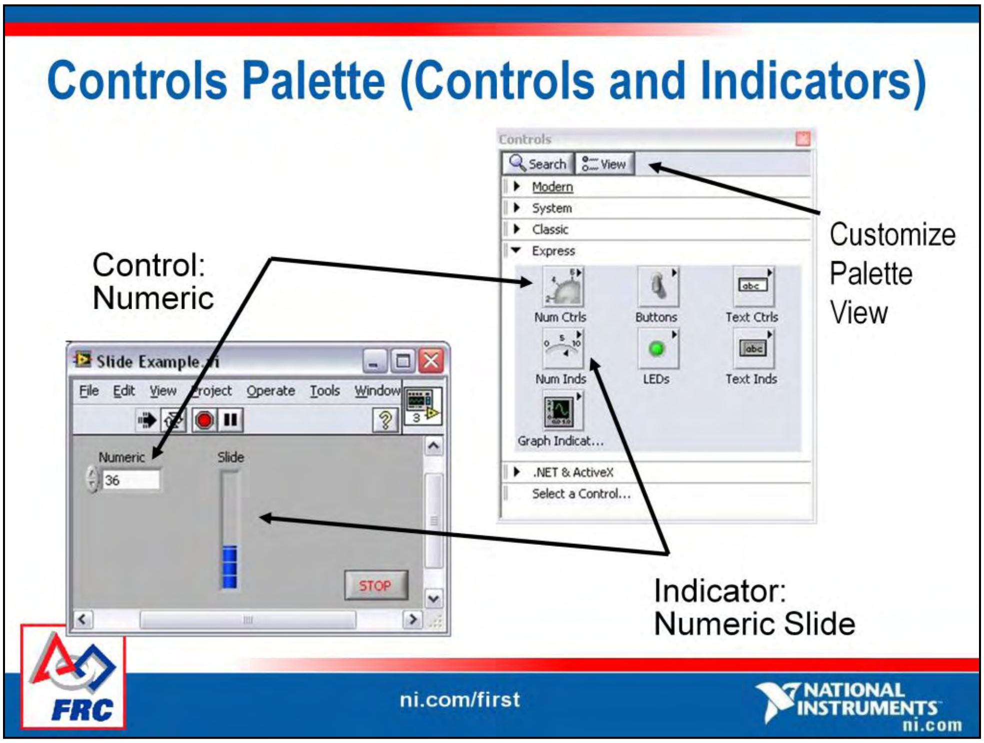
Each VI contains three main parts:

- Front Panel—How the user interacts with the VI.
- Block Diagram—The code that controls the program.
- Icon/Connector—Means of connecting a VI to other VIs.

In LabVIEW, you build a user interface by using a set of tools and objects. The user interface is known as the front panel. You then add code using graphical representations of functions to control the front panel objects. The block diagram contains this code. In some ways, the block diagram resembles a flowchart.

Users interact with the front panel when the program is running. Users can control the program, change inputs, and see data updated in real time. Controls are used for inputs such as adjusting a slide control to set an alarm value, turning a switch on or off, or stopping a program. Indicators are used as outputs. Thermometers, lights, and other indicators display output values from the program. These may include data, program states, and other information.

Every front panel control or indicator has a corresponding terminal on the block diagram. When a VI is run, values from controls flow through the block diagram, where they are used in the functions on the diagram, and the results are passed into other functions or indicators through wires.



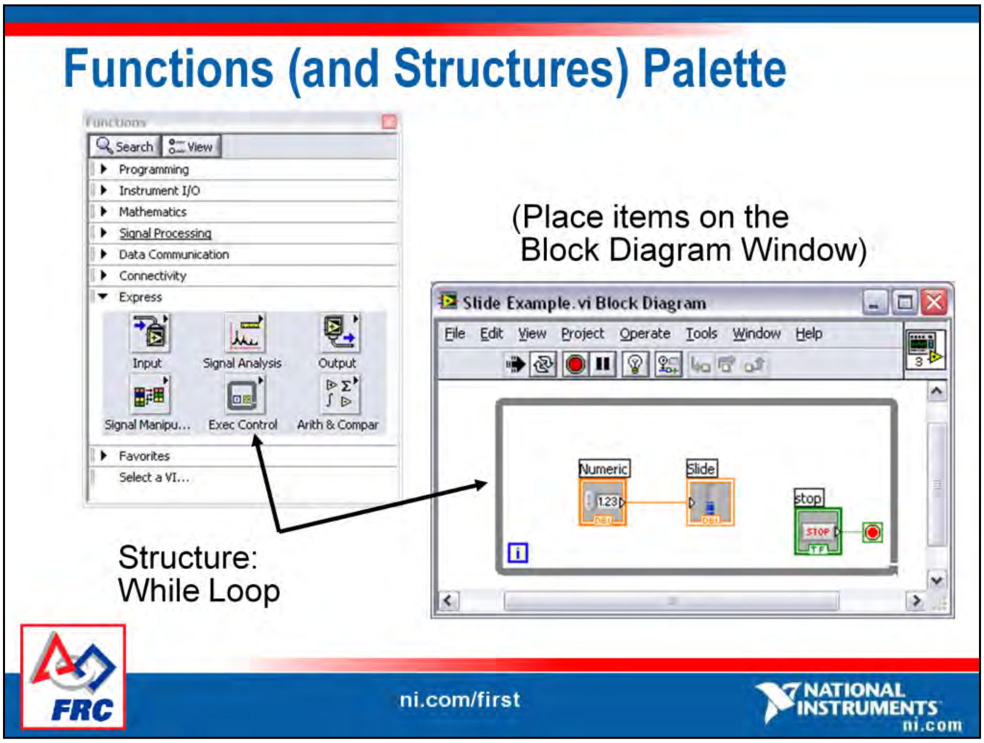
Controls Palette (Controls and Indicators)

Control: Numeric

Customize Palette View

Indicator: Numeric Slide

Use the **Controls** palette to place controls and indicators on the front panel. The **Controls** palette is available only on the front panel. To view the palette, select **Window»Show Controls Palette**. You also can display the **Controls** palette by right-clicking an open area on the front panel. Tack down the **Controls** palette by clicking the pushpin on the top left corner of the palette.



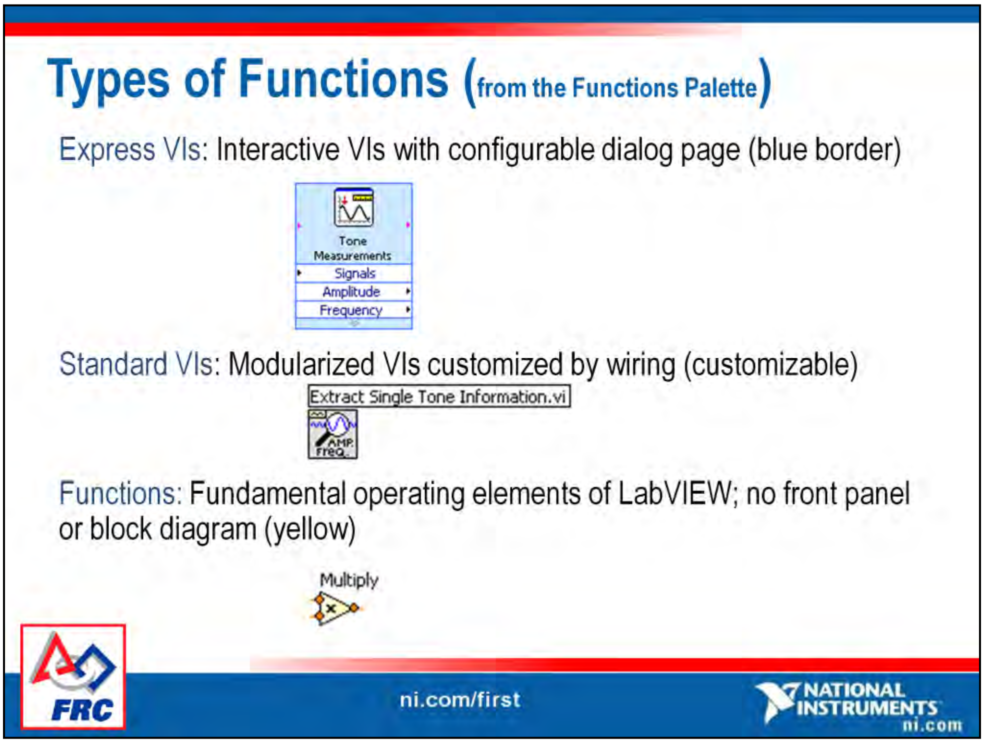
Functions (and Structures) Palette

(Place items on the Block Diagram Window)

Structure: While Loop

ni.com/first

Use the **Functions** palette to build the block diagram. The **Functions** palette is available only on the block diagram. To view the palette, select **Window»Show Functions Palette**. You also can display the **Functions** palette by right-clicking an open area on the block diagram. Tack down the **Functions** palette by clicking the pushpin on the top left corner of the palette.



Types of Functions (from the Functions Palette)

Express VIs: Interactive VIs with configurable dialog page (blue border)

Standard VIs: Modularized VIs customized by wiring (customizable)

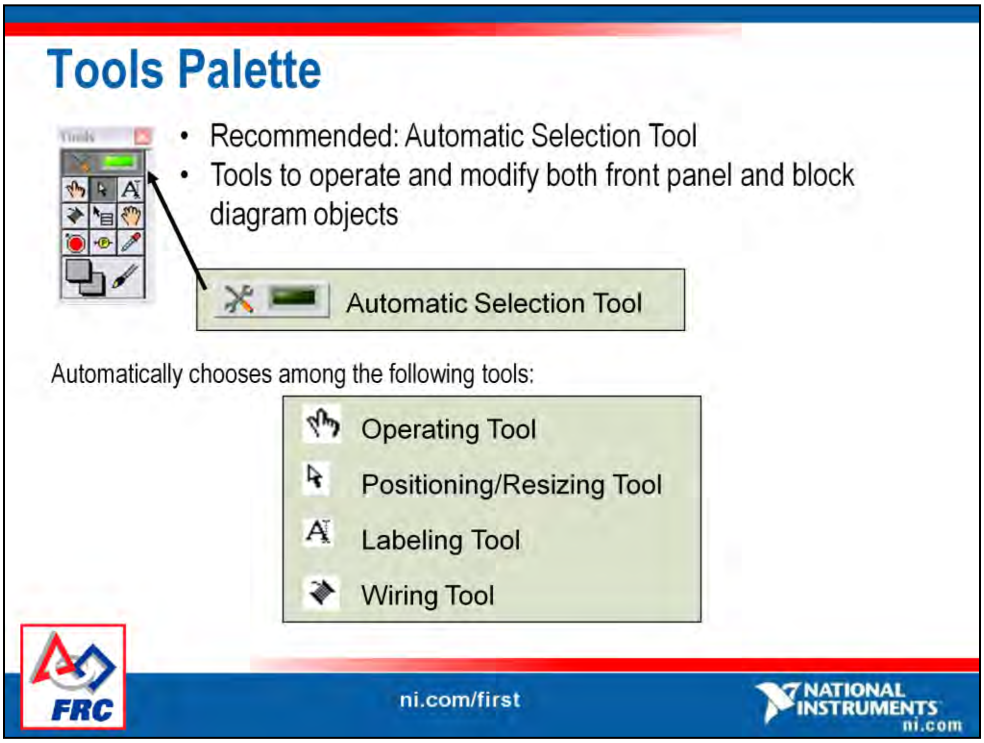
Functions: Fundamental operating elements of LabVIEW; no front panel or block diagram (yellow)

ni.com/first

Express VIs are interactive VIs that have a configuration dialog box that helps the user customize the functionality of the Express VI. LabVIEW then generates a subVI based on these settings.

SubVIs are VIs (consisting of a front panel and a block diagram) that are used within another VI.

Functions are the building blocks of all VIs. Functions do not have a front panel or a block diagram.



Tools Palette

- Recommended: Automatic Selection Tool
- Tools to operate and modify both front panel and block diagram objects

Automatic Selection Tool

Automatically chooses among the following tools:

- Operating Tool
- Positioning/Resizing Tool
- Labeling Tool
- Wiring Tool

ni.com/first

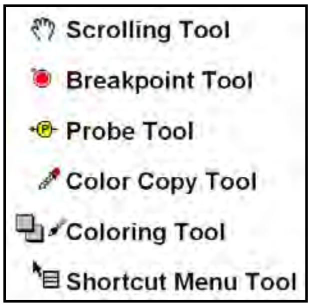
If you enable the automatic selection tool and you move the cursor over objects on the front panel or block diagram, LabVIEW automatically selects the corresponding tool from the **Tools** palette. Toggle the automatic selection tool by clicking the **Automatic Selection Tool** button in the **Tools** palette.

Use the **Operating Tool** to change the values of a control or select the text within a control.

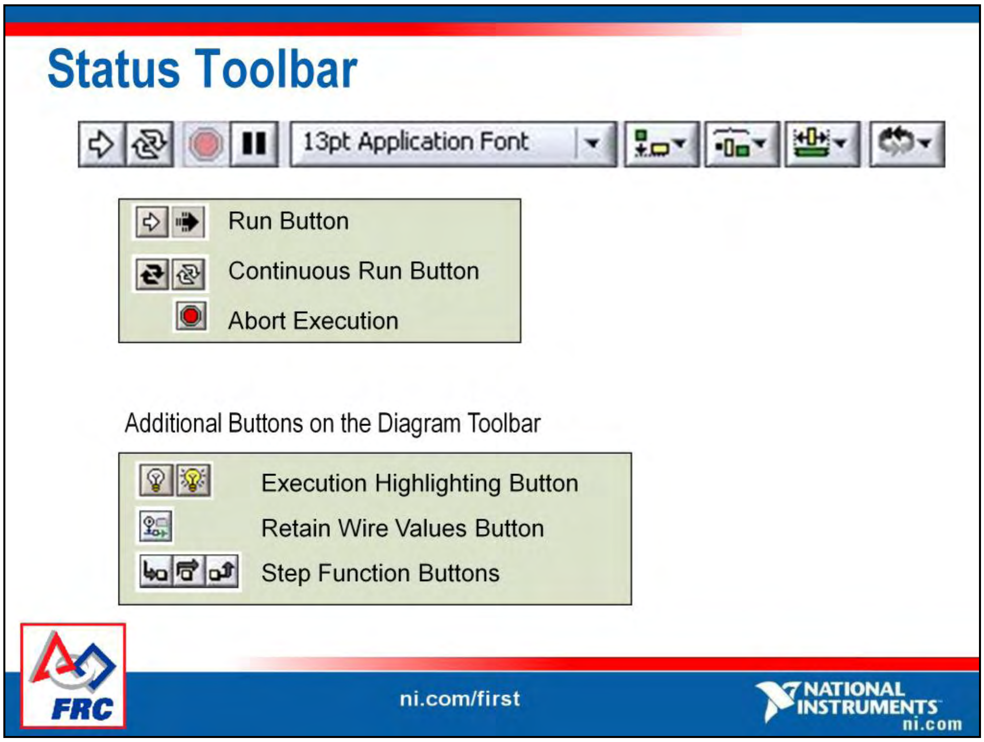
Use the **Positioning Tool** to select, move, or resize objects. The Positioning Tool changes shape when it moves over a corner of a resizable object.

Use the **Labeling Tool** to edit text and create free labels. The Labeling Tool changes to a cursor when you create free labels.

Use the **Wiring Tool** to wire objects together on the block diagram.



- Scrolling Tool
- Breakpoint Tool
- Probe Tool
- Color Copy Tool
- Coloring Tool
- Shortcut Menu Tool



Status Toolbar

Run Button

Continuous Run Button

Abort Execution

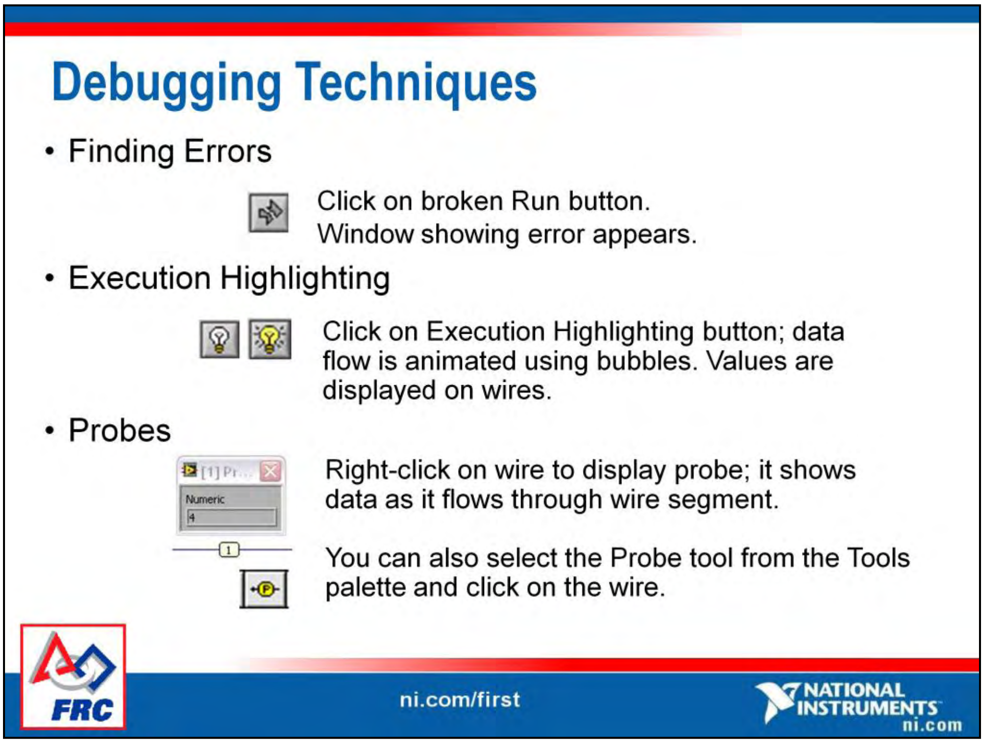
Additional Buttons on the Diagram Toolbar

Execution Highlighting Button

Retain Wire Values Button

Step Function Buttons

- Click the **Run** button to run the VI. While the VI runs, the **Run** button appears with a black arrow if the VI is a top-level VI, meaning it has no callers and therefore is not a subVI.
- Click the **Continuous Run** button to run the VI until you abort or pause it. You also can click the button again to disable continuous running.
- While the VI runs, the **Abort Execution** button appears. Click this button to stop the VI immediately.
  Note: Avoid using the **Abort Execution** button to stop a VI. Either let the VI complete its data flow or design a method to stop the VI programmatically. By doing so, the VI is at a known state. For example, place a button on the front panel that stops the VI when you click it.
- Click the **Pause** button to pause a running VI. When you click the **Pause** button, LabVIEW highlights on the block diagram the location where you paused execution. Click the **Pause** button again to continue running the VI.
- Select the **Text Settings** pull-down menu to change the font settings for the VI, including size, style, and color.
- Select the **Align Objects** pull-down menu to align objects along axes, including vertical, top edge, left, and so on.
- Select the **Distribute Objects** pull-down menu to space objects evenly, including gaps, compression, and so on.
- Select the **Resize Objects** pull-down menu to change the width and height of front panel objects.



Debugging Techniques

- Finding Errors

  Click on broken Run button.
  Window showing error appears.

- Execution Highlighting

  Click on Execution Highlighting button; data flow is animated using bubbles. Values are displayed on wires.

- Probes

  Right-click on wire to display probe; it shows data as it flows through wire segment.

  You can also select the Probe tool from the Tools palette and click on the wire.

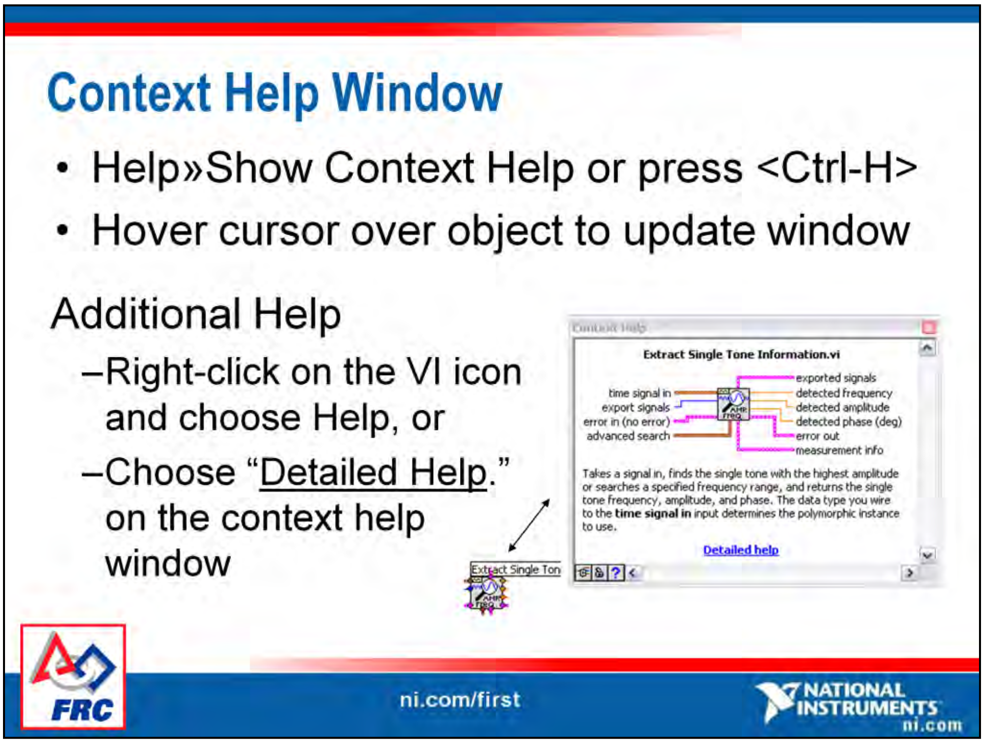
FRC

ni.com/first

NATIONAL INSTRUMENTS
ni.com

When your VI is not executable, a broken arrow is displayed in the Run button in the palette.

- **Finding Errors**: To list errors, click on the broken arrow. To locate the bad object, click on the error message.
- **Execution Highlighting**: Animates the diagram and traces the flow of the data, allowing you to view intermediate values. Click on the **light bulb** on the toolbar.
- **Probe**: Used to view values in arrays and clusters. Click on wires with the **Probe** tool or right-click on the wire to set probes.
- **Retain Wire Values**: Used with probes to view the values from the last iteration of the program.
- **Breakpoint**: Set pauses at different locations on the diagram. Click on wires or objects with the **Breakpoint** tool to set breakpoints.



Context Help Window

- Help»Show Context Help or press <Ctrl-H>
- Hover cursor over object to update window

Additional Help

- –Right-click on the VI icon and choose Help, or
- –Choose “Detailed Help.” on the context help window

Extract Single Tone Information.vi

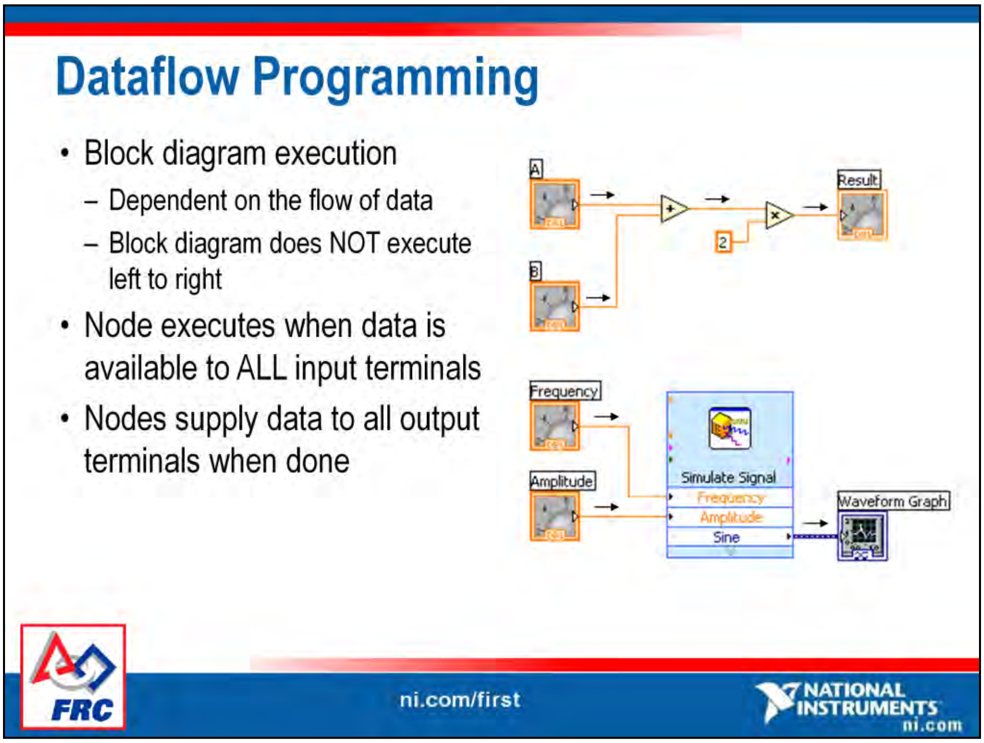
Takes a signal in, finds the single tone with the highest amplitude or searches a specified frequency range, and returns the single tone frequency, amplitude, and phase. The data type you wire to the **time signal in** input determines the polymorphic instance to use.

The **Context Help window** displays basic information about LabVIEW objects when you move the cursor over each object. Objects with context help information include VIs, functions, constants, structures, palettes, properties, methods, events, and dialog box components.

To display the Context Help window, select **Help»Show Context Help**, press the <Ctrl-H> keys, or press the **Show Context Help Window** button in the toolbar.

Connections displayed in Context Help:
Required—bold
Recommended—normal
Optional—dimmed
Additional Help

- **VI, Function, & How-To Help is also available.**
  - **Help» VI, Function, & How-To Help**
  - Right-click the VI icon and choose **Help**, or
  - Choose “**Detailed Help**.” on the context help window.
- **LabVIEW Help—reference style help**
  - **Help»Search the LabVIEW Help…**



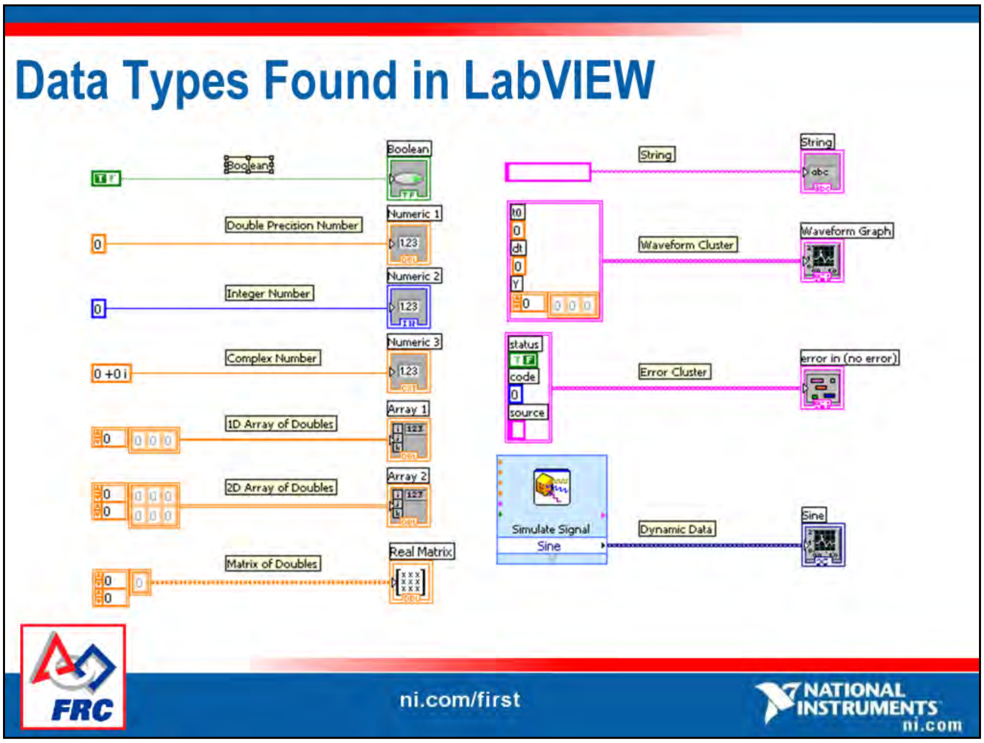
Dataflow Programming

- Block diagram execution
  - Dependent on the flow of data
  - Block diagram does NOT execute left to right
- Node executes when data is available to ALL input terminals
- Nodes supply data to all output terminals when done

LabVIEW follows a dataflow model for running VIs. A block diagram node executes when all its inputs are available. When a node completes execution, it supplies data to its output terminals and passes the output data to the next node in the dataflow path. Visual Basic, C++, JAVA, and most other text-based programming languages follow a control flow model of program execution. In control flow, the sequential order of program elements determines the execution order of a program.

Consider the block diagram above. It adds two numbers and then multiplies by 2 from the result of the addition. In this case, the block diagram executes from left to right, not because the objects are placed in that order, but because one of the inputs of the Multiply function is not valid until the Add function has finished executing and passed the data to the Multiply function. Remember that a node executes only when data is available at all of its input terminals, and it supplies data to its output terminals only when it finishes execution. In the second piece of code, the Simulate Signal Express VI receives input from the controls and passes its result to the Graph.

You may consider the add-multiply and the simulate signal code to co-exist on the same block diagram in parallel. This means that they will both begin executing at the same time and run independently of one another. If the computer running this code had multiple processors, these two pieces of code could run independently of one another (each on its own processor) without any additional coding.



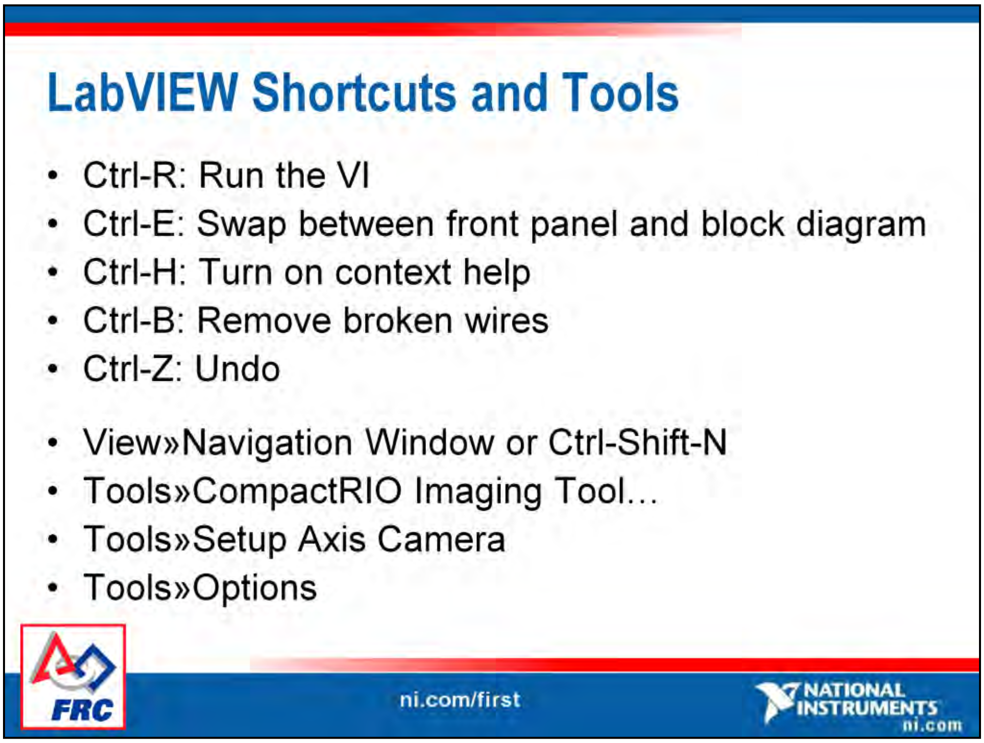
Data Types Found in LabVIEW

LabVIEW uses many common data types, including Boolean, numeric, arrays, strings, and clusters. The color and symbol of each terminal indicate the data type of the control or indicator. Control terminals have a thicker border than indicator terminals. Also, arrows appear on front panel terminals to indicate whether the terminal is a control or an indicator. An arrow appears on the right if the terminal is a control and on the left if the terminal is an indicator.

Definitions

- **Array:** Arrays group data elements of the same type. An array consists of elements and dimensions. Elements are the data that make up the array, and a dimension is the length, height, or depth of an array. An array can have one or more dimensions and as many as (2^{31}) – 1 elements per dimension, memory permitting.
- **Cluster:** Clusters group data elements of mixed types, such as a bundle of wires in a telephone cable, where each wire in the cable represents a different element of the cluster.

See **Help»Search the LabVIEW Help…** for more information. The *LabVIEW User Manual* on `ni.com` provides additional references for data types found in LabVIEW.



LabVIEW Shortcuts and Tools

- Ctrl-R: Run the VI
- Ctrl-E: Swap between front panel and block diagram
- Ctrl-H: Turn on context help
- Ctrl-B: Remove broken wires
- Ctrl-Z: Undo

- View»Navigation Window or Ctrl-Shift-N
- Tools»CompactRIO Imaging Tool…
- Tools»Setup Axis Camera
- Tools»Options

FRC
ni.com/first
NATIONAL INSTRUMENTS
ni.com

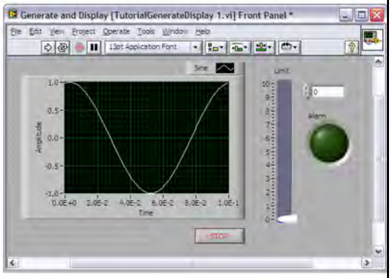
•**View»Navigation Window or Ctrl-Shift-N** The navigation window can help you navigate through your program if you have a large block diagram.

•**Tools»CompactRIO Imaging Tool…** This is how to get to the Imaging Tool to update the firmware on your cRIO-FRC.

•**Tools»Setup Axis Camera** This setup is used to configure your Axis camera including the IP address.

•**Tools»Options** Use this for the myriad options on setting up LabVIEW, customizing the environment to your liking, and so on.

LabVIEW Exercise 1: Create a Simple LabVIEW VI



You will learn

- How to work in the LabVIEW environment

Create an application that

- Simulates data collection
- Turns on a front panel indicator when threshold is reached

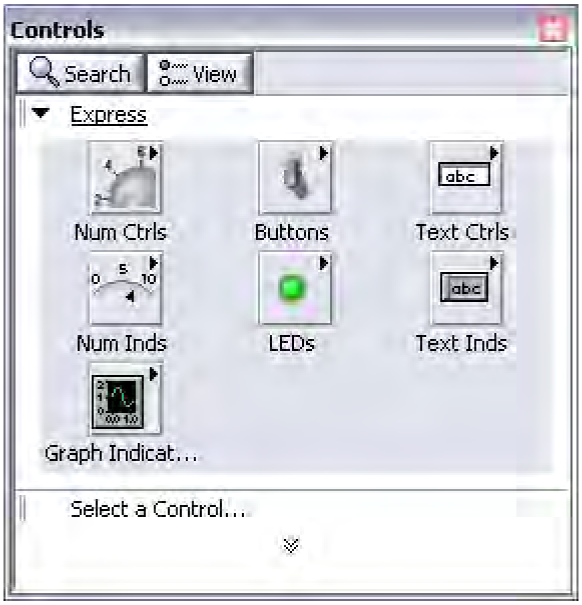
ni.com/first

Exercise 1: Create a Simple LabVIEW VI

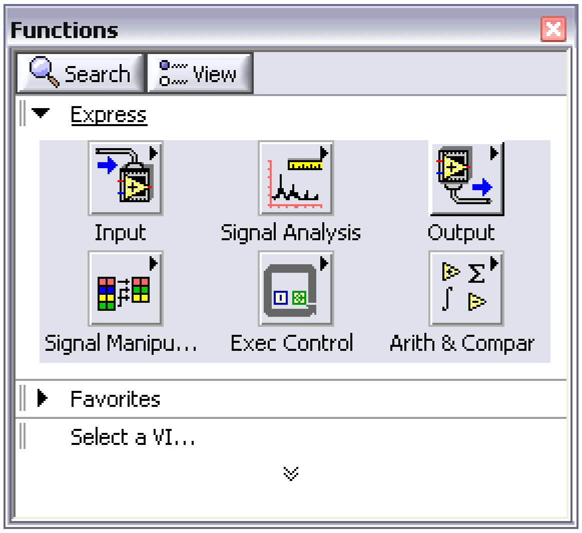
Use this exercise to create a simple LabVIEW VI that simulates an analog signal and plots it on a waveform graph. The VI tests the input values against a user-specified limit and lights an LED if the input value exceeds that limit.

Below are pictures identifying each of the palettes found in LabVIEW to assist you as you complete these exercises.

Note: LabVIEW has a built-in Automatic Tool Selection feature that changes the behavior of the cursor depending on what type of object you are pointing at.

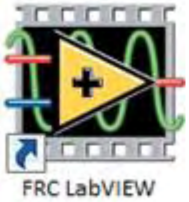


Controls Palette

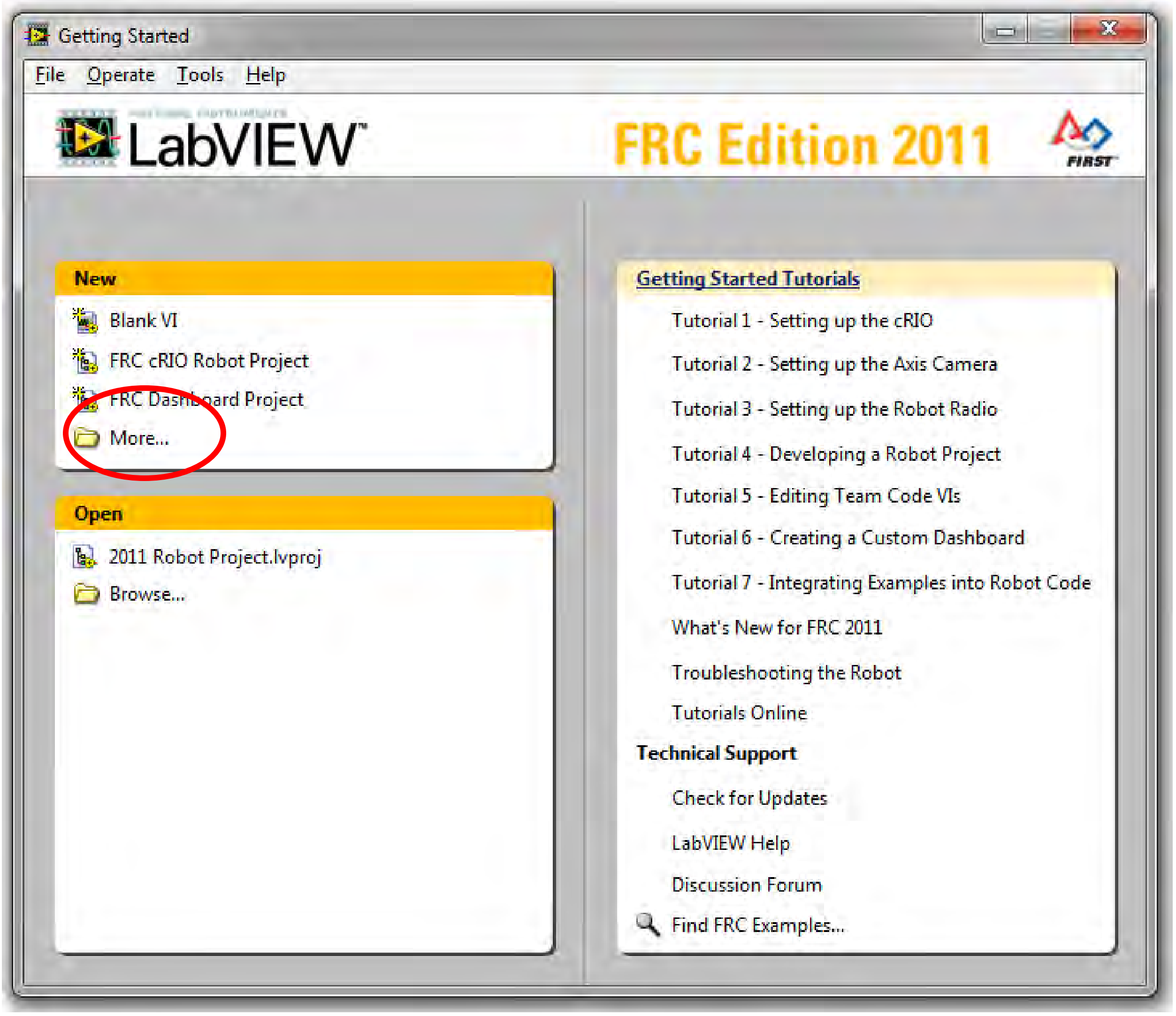


Functions Palette

1. If you have not already done so, click the **LabVIEW** icon on your quick launch toolbar. If the icon is not present in the quick launch, the desktop should feature a LabVIEW shortcut or you can navigate to Start»Programs»National Instruments»LabVIEW 8.6»LabVIEW.



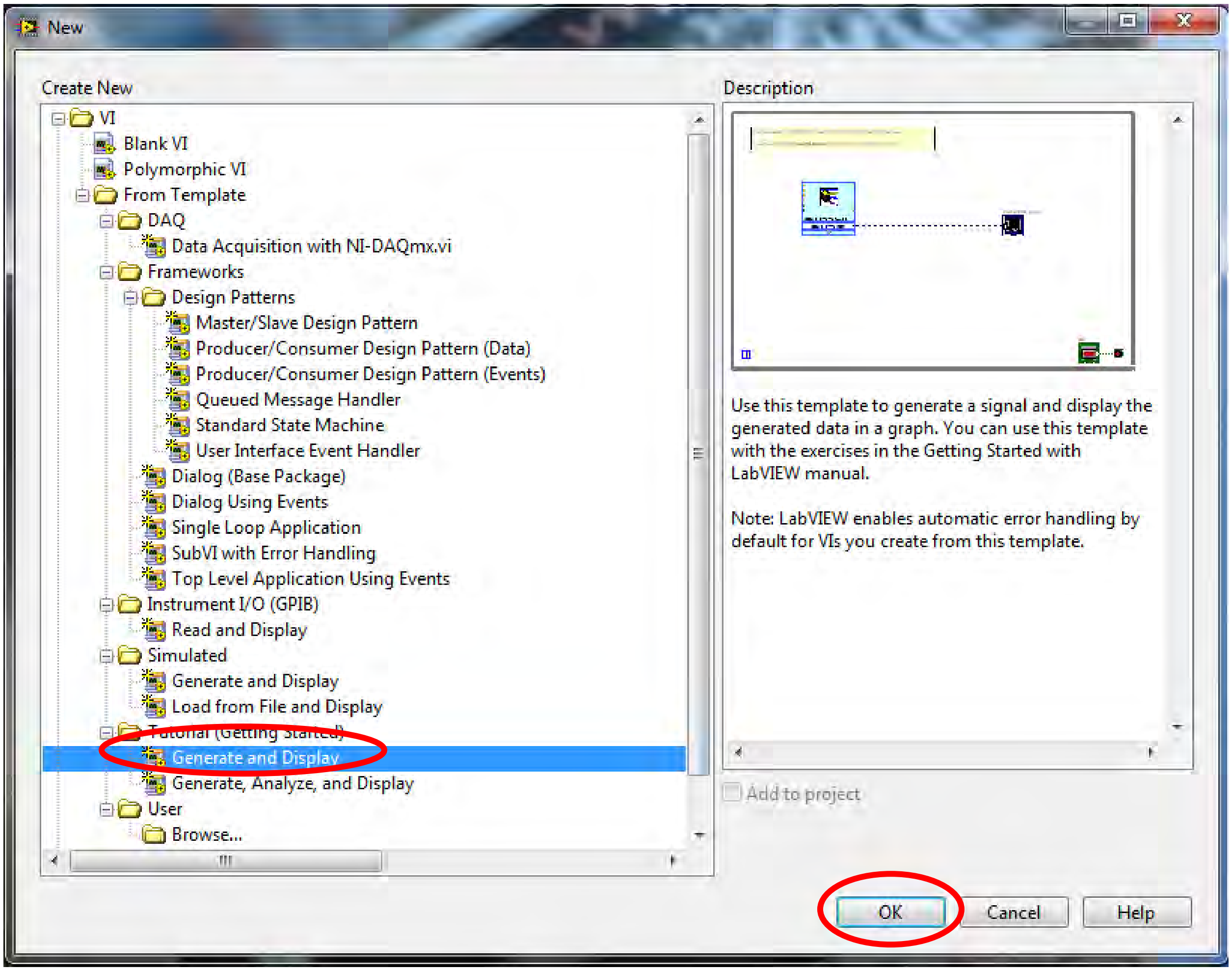
Once you launch LabVIEW, a splash screen like the following appears.


Getting Started
File Operate Tools Help
LabVIEW
FRC Edition 2011
FIRST
New
Blank VI
FRC cRIO Robot Project
FRC Dashboard Project
More...
Open
2011 Robot Project.lvproj
Browse...
Getting Started Tutorials
Tutorial 1 - Setting up the cRIO
Tutorial 2 - Setting up the Axis Camera
Tutorial 3 - Setting up the Robot Radio
Tutorial 4 - Developing a Robot Project
Tutorial 5 - Editing Team Code VIs
Tutorial 6 - Creating a Custom Dashboard
Tutorial 7 - Integrating Examples into Robot Code
What's New for FRC 2011
Troubleshooting the Robot
Tutorials Online
Technical Support
Check for Updates
LabVIEW Help
Discussion Forum
Find FRC Examples...

2. Click **More…**

3. Under the VI tree heading expand **From Template**. Notice the different categories on the left side of the window that correspond to the types of tasks from which you can choose. You can select **Blank VI** to start from scratch. You also can use template VIs as a starting point to build your application. **Projects** and **Other Files** are more advanced components that are not described in detail in this manual. For more information on any of the listings in the **New** Dialog Box, click the **Help** button in the lower right corner of the window.

4. Select **VI»From Template»Tutorial (Getting Started)»Generate and Display** and click **OK**.

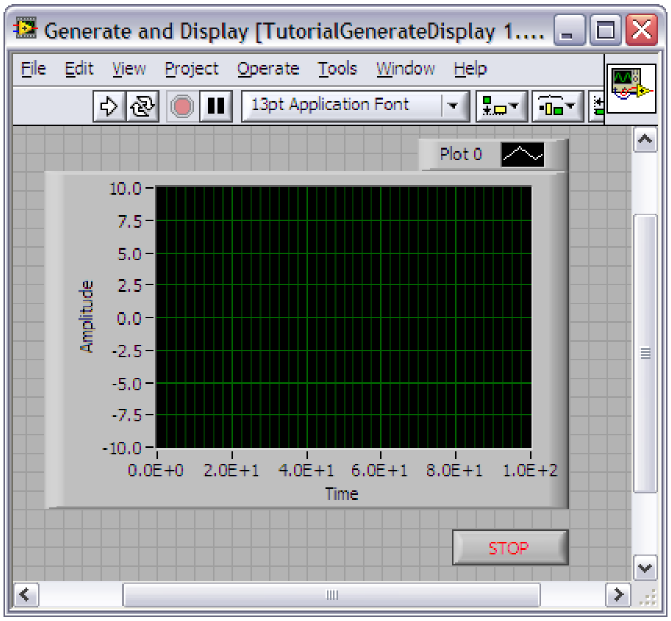


Use this template to generate a signal and display the generated data in a graph. You can use this template with the exercises in the Getting Started with LabVIEW manual.

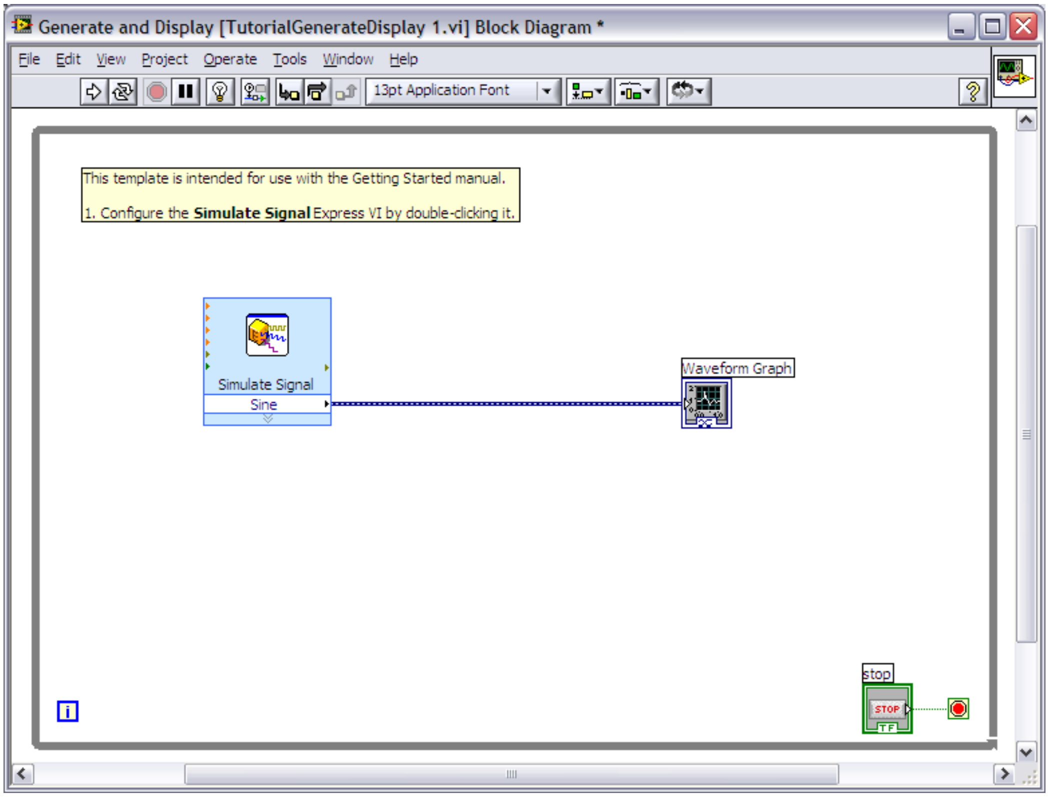
Note: LabVIEW enables automatic error handling by default for VIs you create from this template.

Two windows appear. The gray window is the front panel and the white one is the block diagram. The front panel contains the parts of your VI used for presenting information while interfacing with the user, and the block diagram contains the code that controls the VI functionality. You can toggle between the two windows by selecting **Window»Show Block Diagram** or **Window»Show Front Panel**. You can also switch between the windows by pressing <Ctrl-E> on the keyboard.

5. Examine the front panel and block diagram of this template VI in the below figure. The front panel contains a waveform chart and a **STOP** button.



The block diagram contains a Simulate Signal VI that is configured to simulate a sine wave and plot it to the graph.


Generate and Display [TutorialGenerateDisplay 1.vi] Block Diagram *
File Edit View Project Operate Tools Window Help
13pt Application Font
Simulate Signal
Sine
Waveform Graph
stop
STOP
TF
i

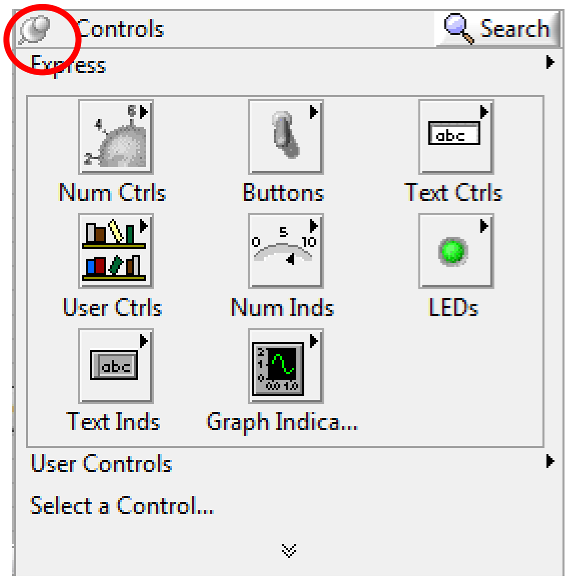
This template is intended for use with the Getting Started manual.

1. Configure the **Simulate Signal** Express VI by double-clicking it.

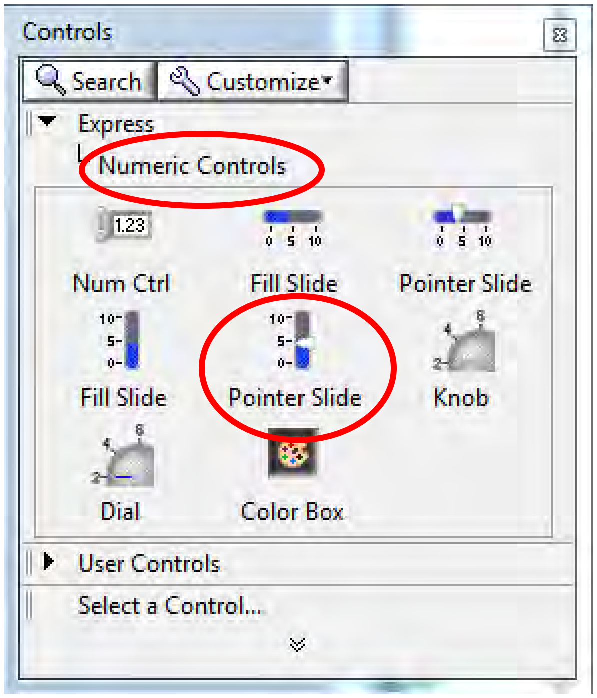
6. Switch back to the front panel by pressing <Ctrl-E>. Since the **Run** button (the white arrow in the top left corner) is solid, you can run this VI as it is. Click the **Run** button and examine the operation of the VI. When you are finished, click the **STOP** button on the front panel to stop running the VI.

   Note: As you will see later in the exercise, when the **Run** button in the upper left corner of both the front panel and the block diagram changes from a solid white arrow to a broken gray arrow, this new icon indicates that the VI is currently not executable.

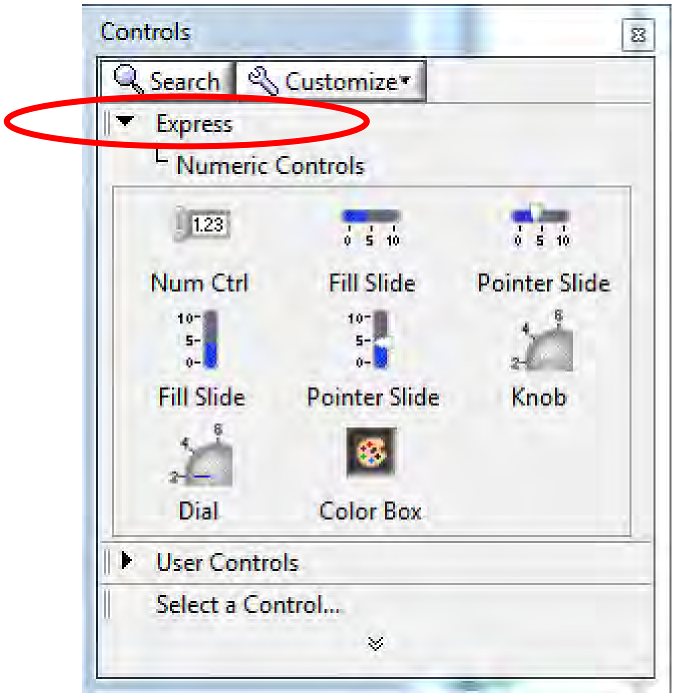
7. Now you can add some functionality to this basic VI. Modify the VI to flash an alarm whenever the signal value is above a certain level. Open the **Controls** palette (if it is not open already) by right-clicking empty space on the front panel window. A small pushpin icon in the upper left corner of this palette appears. Click this pushpin to force the palette to remain on your screen.



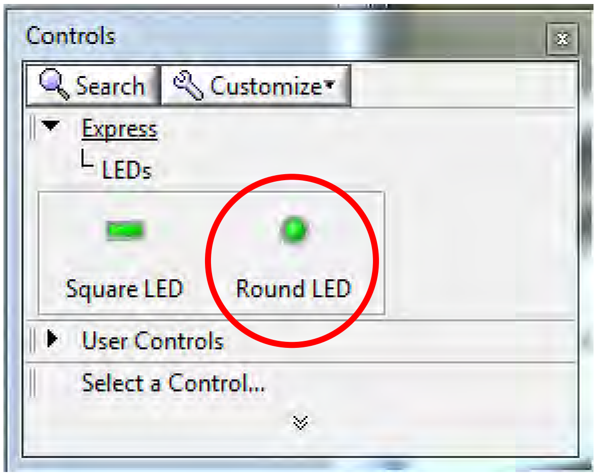
8. Click the **Numeric Controls** sub-palette and select a **Vertical Pointer Slide** to be placed on the front panel. To do this, click the **Vertical Pointer Slide** and drag it to the front panel. Click once to place it.



9. Click the **Express** menu item on the **Controls** palette to return to the Express Controls palette.



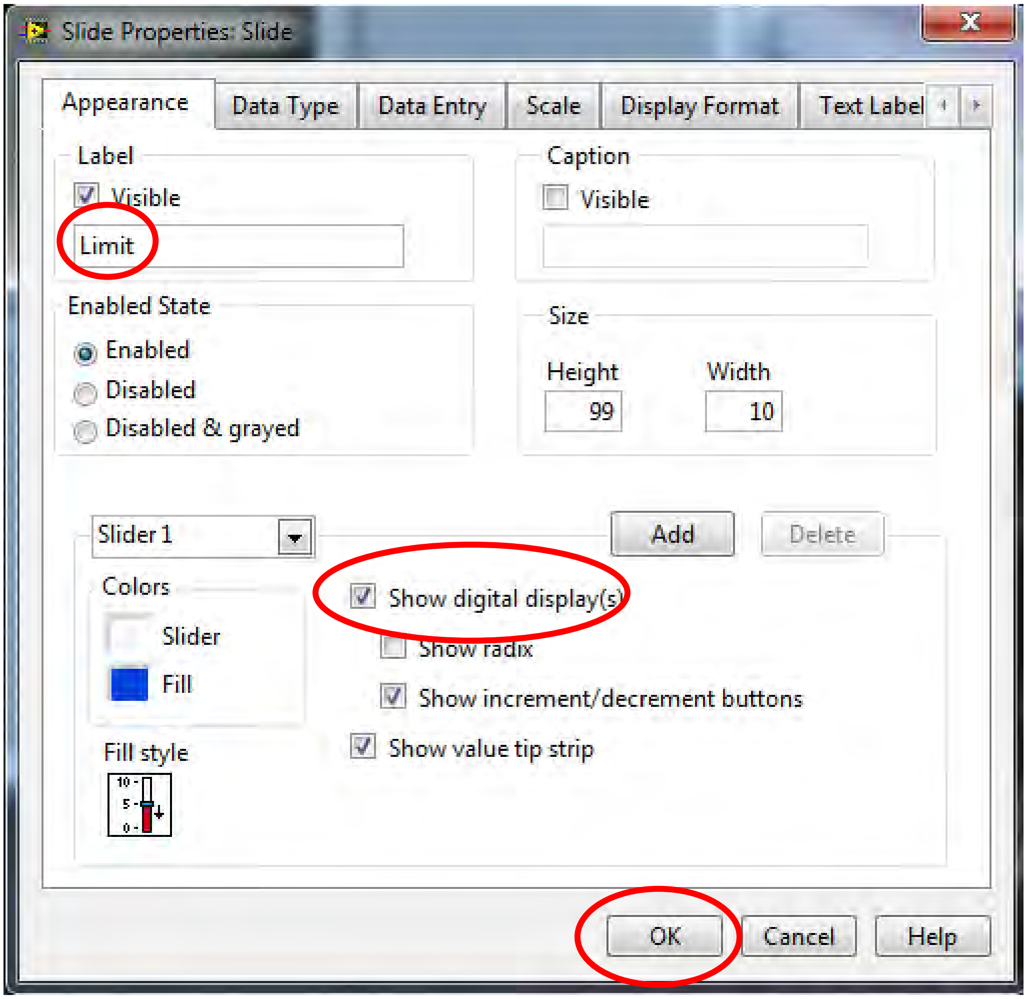
10. Click the **LEDs** sub-palette and place a **Round LED** on the front panel.



11. Right-click the **Vertical Pointer Slide** and select **Properties**. The Properties Dialog Box appears. Examine the different properties that you can modify. Make the following changes on the **Appearance** tab and click **OK** to apply the changes.

    Label: `Limit`

    Slider 1: Check Show digital display(s)



Label
☑ Visible
Limit

Caption
□ Visible

Enabled State
● Enabled
○ Disabled
○ Disabled & grayed

Size
Height 99
Width 10

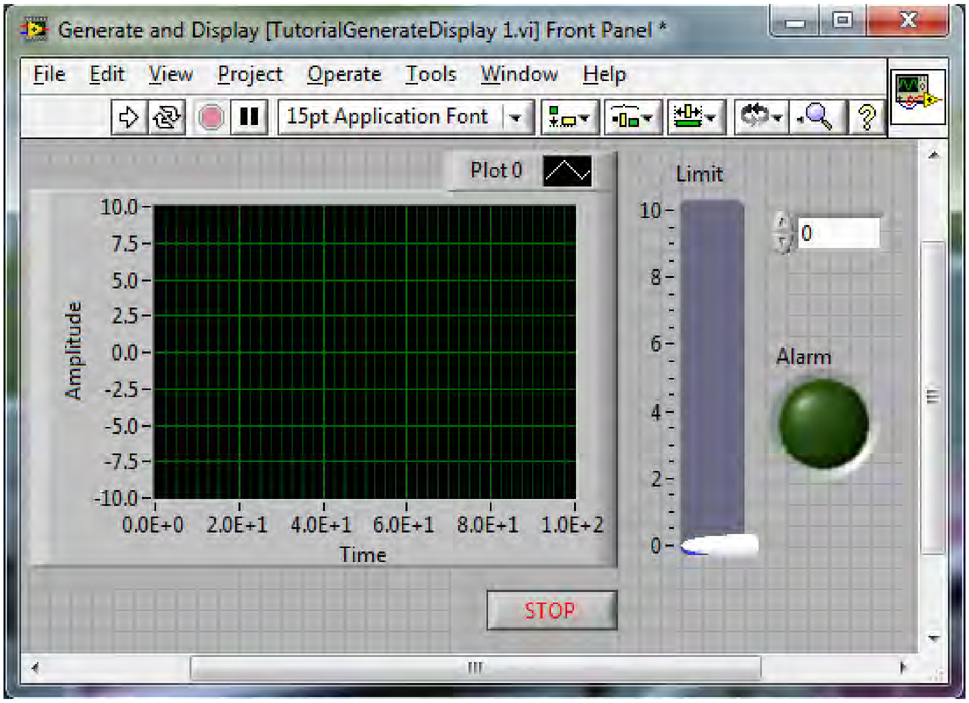
Slider 1

Colors
Slider
Fill 🟦

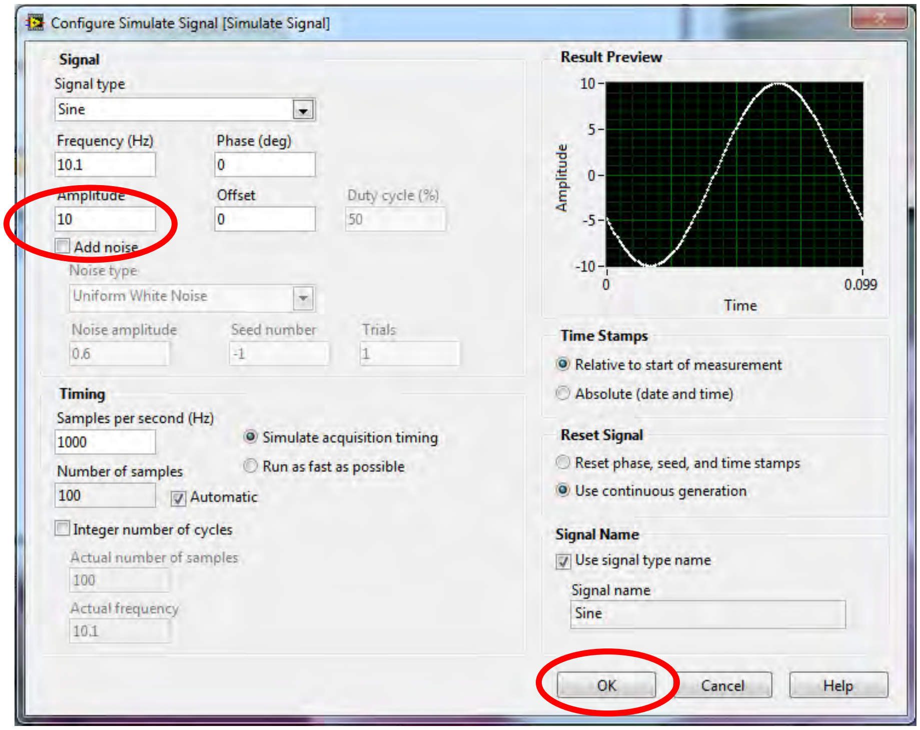
Fill style

☑ Show digital display(s)
□ Show radix
☑ Show increment/decrement buttons
☑ Show value tip strip

12. Right-click the Round LED labeled **Boolean** and select **Properties**. Examine the different properties that you can modify. On the **Appearance** tab, change the label from `Boolean` to **`Alarm`**. Click **OK** to apply your change. Move the objects on the front panel so it resembles the following.


Generate and Display [TutorialGenerateDisplay 1.vi] Front Panel *
File Edit View Project Operate Tools Window Help
15pt Application Font
Plot 0
Amplitude
10.0
7.5
5.0
2.5
0.0
-2.5
-5.0
-7.5
-10.0
0.0E+0 2.0E+1 4.0E+1 6.0E+1 8.0E+1 1.0E+2
Time
Limit
10
8
6
4
2
0
Alarm
STOP

13. Switch to the block diagram by pressing <Ctrl-E>. Double-click the **Simulate Signal** ExpressVI to bring up its properties window. Examine the different properties you can modify. Change the **Amplitude** of the signal to **10**. Click **OK** to apply this change and to close the properties window.


Configure Simulate Signal [Simulate Signal]

Signal

Signal type: Sine

Frequency (Hz): 10.1

Phase (deg): 0

Amplitude: 10

Offset: 0

Duty cycle (%): 50

□ Add noise

Noise type: Uniform White Noise

Noise amplitude: 0.6

Seed number: -1

Trials: 1

Timing

Samples per second (Hz): 1000

● Simulate acquisition timing

○ Run as fast as possible

Number of samples: 100

☑ Automatic

□ Integer number of cycles

Actual number of samples: 100

Actual frequency: 10.1

Result Preview

Time Stamps

● Relative to start of measurement

○ Absolute (date and time)

Reset Signal

○ Reset phase, seed, and time stamps

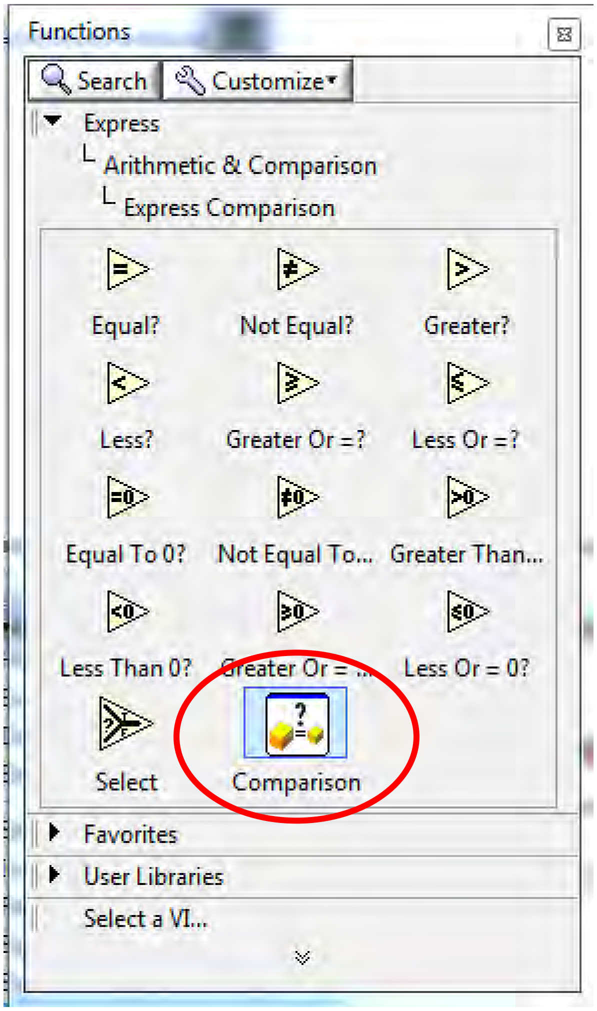
● Use continuous generation

Signal Name

☑ Use signal type name

Signal name: Sine

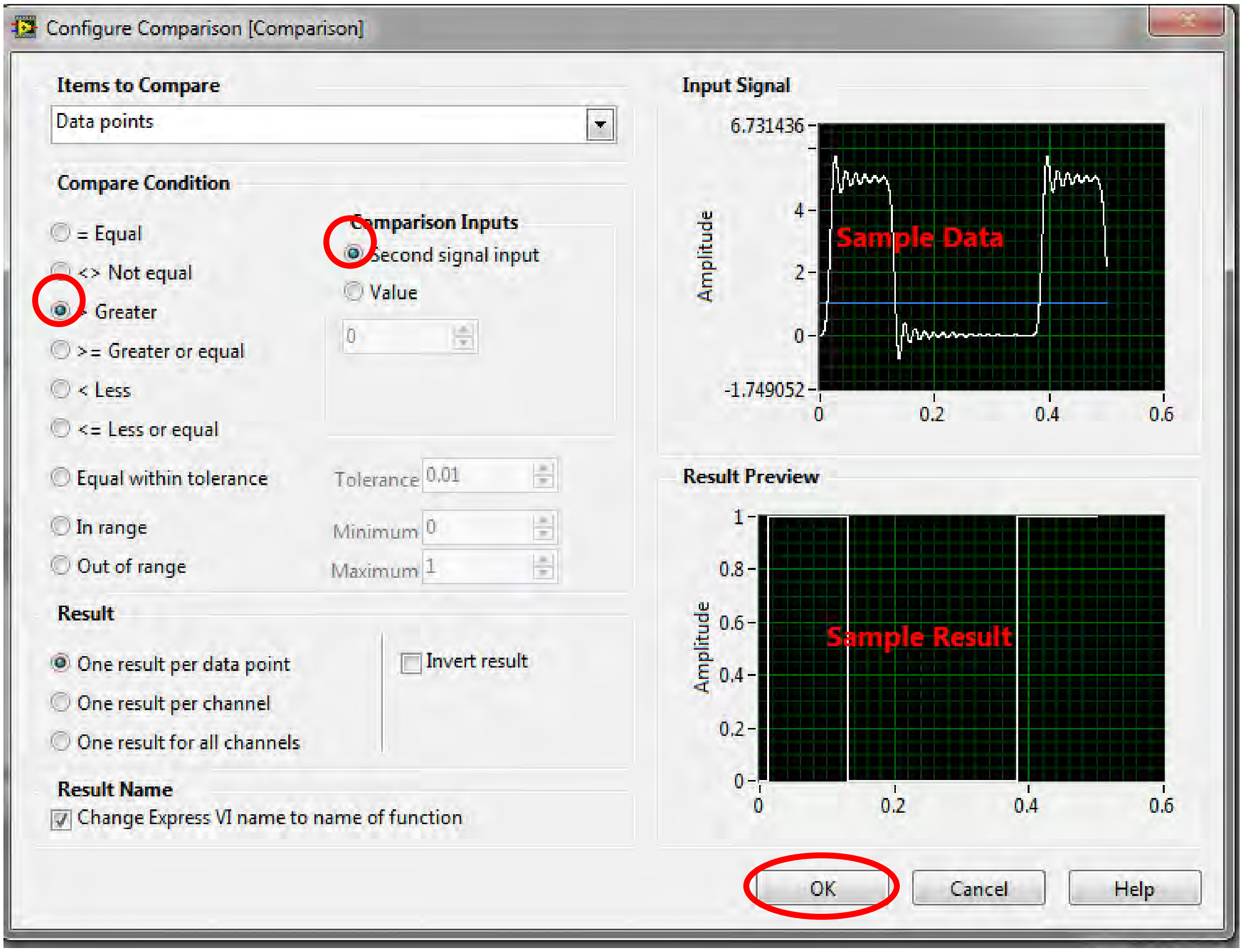
14. Open the functions palette by right-clicking the block diagram. Select **Arithmetic & Comparison»Comparison** and place the **Comparison** Express VI within the while loop on the diagram.



When you place the Comparison Express VI on the block diagram, a dialog box appears that you can use to configure which type of comparison you want to implement. Make the following selections and then click **OK** to apply these changes and to close the dialog box.

Compare Condition: Greater

Comparison Inputs: Compare to second signal input



Configure Comparison [Comparison]

Items to Compare

Data points

Compare Condition

○ = Equal

○ <> Not equal

● > Greater

○ >= Greater or equal

○ < Less

○ <= Less or equal

○ Equal within tolerance Tolerance 0.01

○ In range Minimum 0

○ Out of range Maximum 1

Comparison Inputs

● Second signal input

○ Value

0

Result

● One result per data point

○ One result per channel

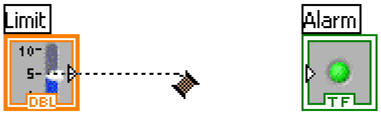
○ One result for all channels

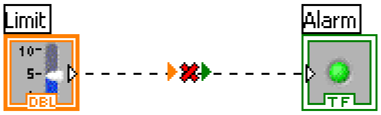
□ Invert result

Result Name

☑ Change Express VI name to name of function

15. You can connect controls, functions, and indicators on the block diagram by pointing to an object and clicking it when the cursor changes to a spool of wire. You can then move the cursor to the object you want to connect it to and click again. Connect the **Limit** control to the **Alarm** indicator.

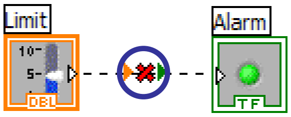




Note: The **Run** button in the upper left corner of both the front panel and the block diagram has changed from a solid white arrow to a broken gray arrow. This new icon indicates that the VI is currently not executable. If you click the **Run** button when it is solid and white, it runs the VI. Clicking it when it is broken and gray opens the error list dialog box to help you debug the VI.

16. Click the **Run** button now. The resulting dialog box shows that, in this case, the error results from connecting the terminals of two different types. Since the **Limit** control is a Numeric type and the **Alarm** indicator is a Boolean type, you cannot wire these two terminals together. Highlight the error by clicking it and then click **Show Error**. LabVIEW highlights the location of the error.

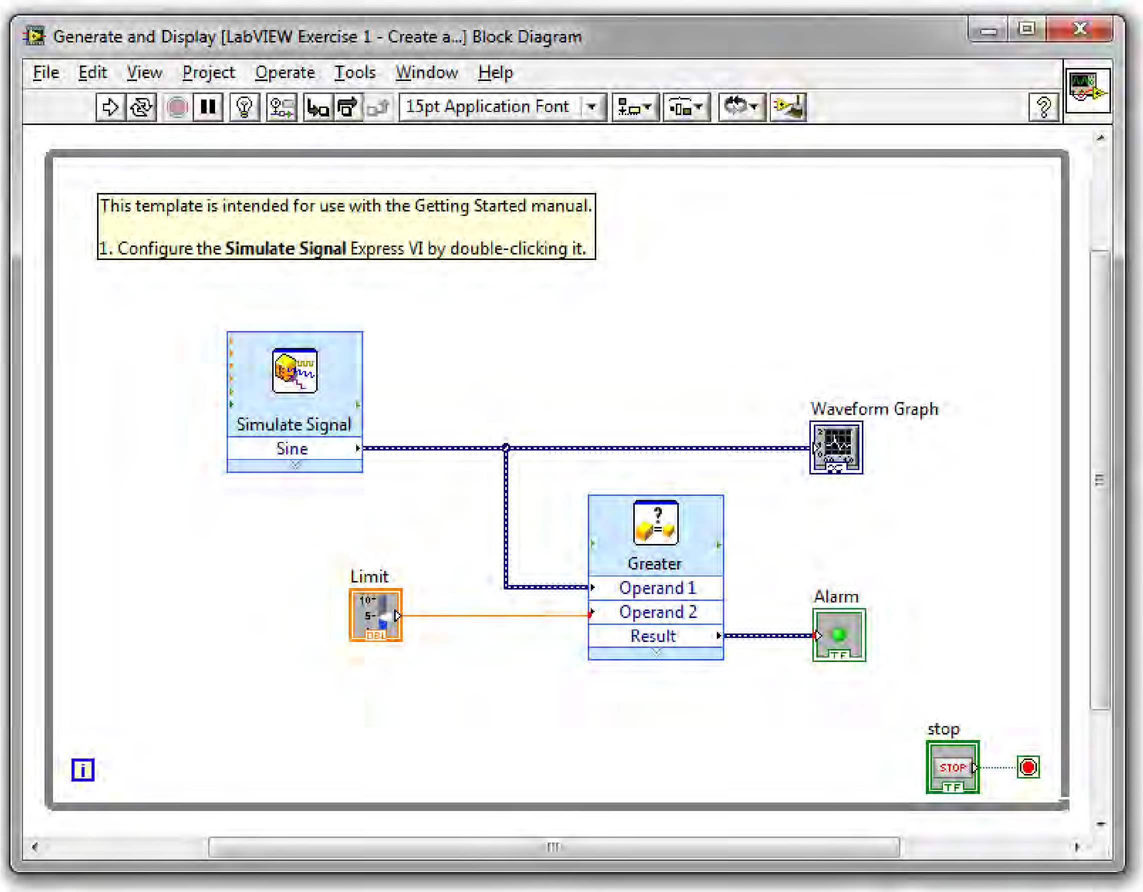
17. Notice that the wire between **Limit** and **Alarm** is dashed and a red × is displayed on it.



To delete this broken wire, press <Ctrl-B>. This keyboard shortcut removes all broken wires from the block diagram.

18. Make your block diagram resemble the below image by completing the following steps.
    a. Wire the **Limit** control to the **Operand 2** input of the **Comparison** function.
    b. Connect the wire between the **Simulate Signals** block and the **Waveform Graph** to the **Operand 1** input of the **Comparison** function.
    c. Wire the **Result** output of the **Comparison** function to the **Alarm** indicator.

    Your block diagram should now resemble the following:


Generate and Display [LabVIEW Exercise 1 - Create a...] Block Diagram
File Edit View Project Operate Tools Window Help
15pt Application Font
Simulate Signal
Sine
Waveform Graph
Greater
Operand 1
Operand 2
Result
Limit
Alarm
stop
STOP

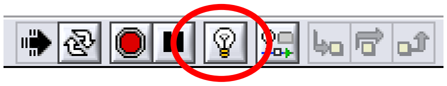
This template is intended for use with the Getting Started manual.

1. Configure the **Simulate Signal** Express VI by double-clicking it.

19. Switch to the front panel by pressing <Ctrl-E>.

20. **Run** the VI. While running the VI, you can change the **Limit** value by moving the slider or changing the digital display next to it. Also notice that when a data point received from the **Simulate Signal VI** is greater than the **Limit** value, the **Alarm** indicator lights up.

    While the VI is still running, switch to the block diagram by pressing <Ctrl-E>. Enable **Highlight Xxecution** by clicking the light bulb on the tool bar. This slow the program and allows you to see the flow of data through your program.



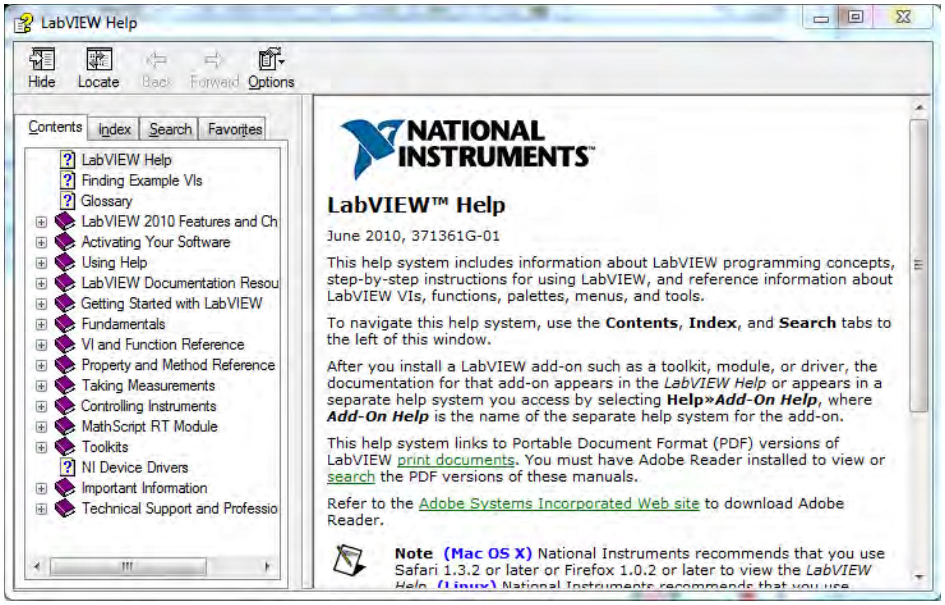
21. When you are finished, stop the VI by clicking the **STOP** button on the front panel.

    END LABVIEW EXERCISE 1.

Optional Exercise 1: The LabVIEW Help System

The LabVIEW help system is a great place to learn about LabVIEW and visit when you have questions. This exercise introduces you to this rich source of information.

1. Go back to the VI you just created and press <F1> on the keyboard to start the help system.



LabVIEW™ Help

June 2010, 371361G-01

This help system includes information about LabVIEW programming concepts, step-by-step instructions for using LabVIEW, and reference information about LabVIEW VIs, functions, palettes, menus, and tools.

To navigate this help system, use the **Contents**, **Index**, and **Search** tabs to the left of this window.

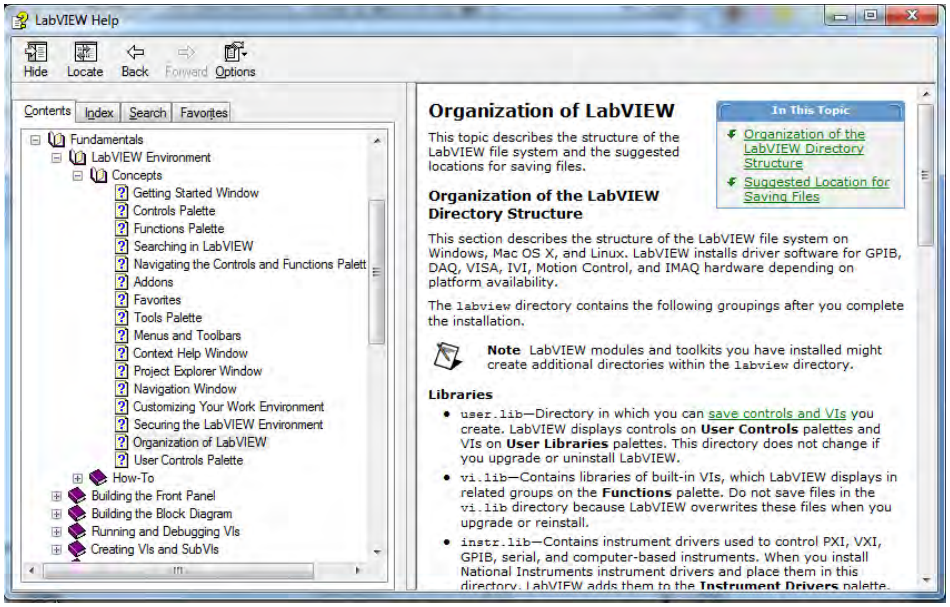
After you install a LabVIEW add-on such as a toolkit, module, or driver, the documentation for that add-on appears in the *LabVIEW Help* or appears in a separate help system you access by selecting **Help»*Add-On Help***, where ***Add-On Help*** is the name of the separate help system for the add-on.

This help system links to Portable Document Format (PDF) versions of LabVIEW print documents. You must have Adobe Reader installed to view or search the PDF versions of these manuals.

Refer to the Adobe Systems Incorporated Web site to download Adobe Reader.

Note **(Mac OS X)** National Instruments recommends that you use Safari 1.3.2 or later or Firefox 1.0.2 or later to view the *LabVIEW Help*. **(Linux)** National Instruments recommends that you use

2. Expand **Fundamentals»LabVIEW Environment** and explore the information available here, look around, and get a feel for how it is organized.



Organization of LabVIEW

This topic describes the structure of the LabVIEW file system and the suggested locations for saving files.

Organization of the LabVIEW Directory Structure

This section describes the structure of the LabVIEW file system on Windows, Mac OS X, and Linux. LabVIEW installs driver software for GPIB, DAQ, VISA, IVI, Motion Control, and IMAQ hardware depending on platform availability.

The `labview` directory contains the following groupings after you complete the installation.

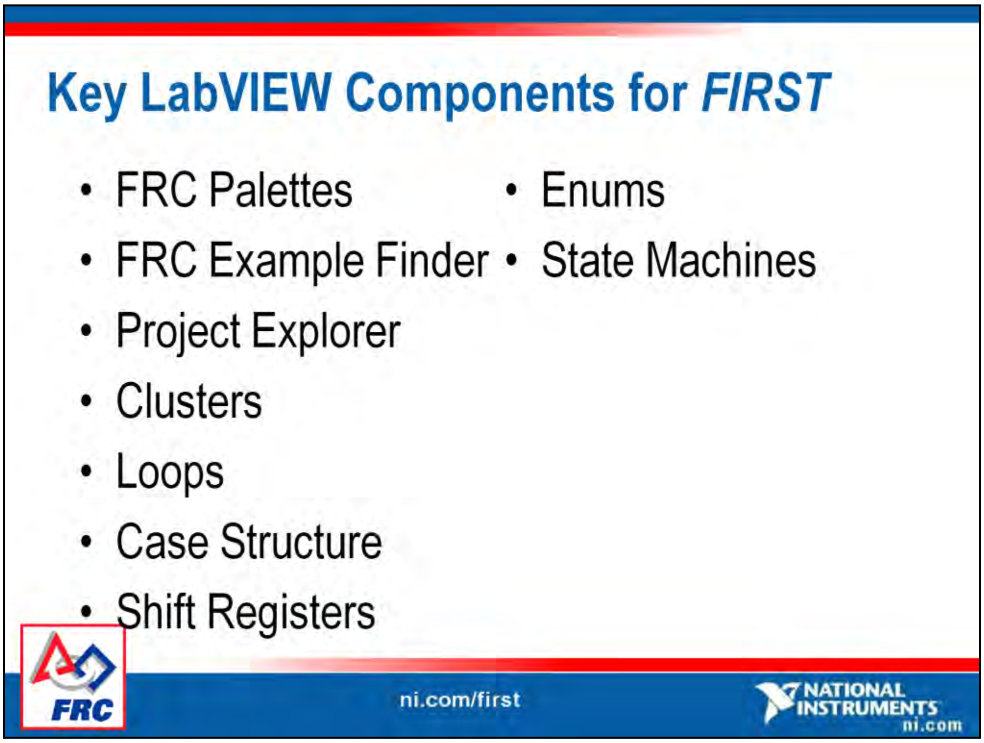
Note LabVIEW modules and toolkits you have installed might create additional directories within the `labview` directory.

Libraries

- `user.lib`—Directory in which you can save controls and VIs you create. LabVIEW displays controls on **User Controls** palettes and VIs on **User Libraries** palettes. This directory does not change if you upgrade or uninstall LabVIEW.
- `vi.lib`—Contains libraries of built-in VIs, which LabVIEW displays in related groups on the **Functions** palette. Do not save files in the `vi.lib` directory because LabVIEW overwrites these files when you upgrade or reinstall.
- `instr.lib`—Contains instrument drivers used to control PXI, VXI, GPIB, serial, and computer-based instruments. When you install National Instruments instrument drivers and place them in this directory, LabVIEW adds them to the **Instrument Drivers** palette.

3. Take a few minutes to explore other topics in the help system.

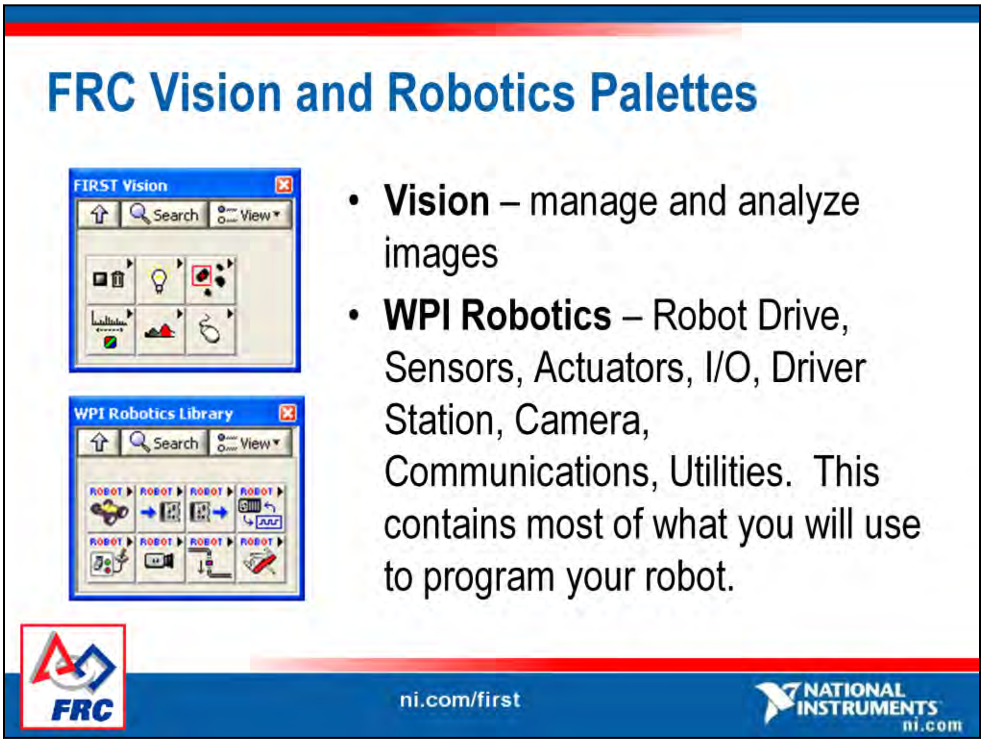
4. Click on the **Search** tab to try searching on analysis functions for features you might need in your work applications.



Key LabVIEW Components for *FIRST*

- FRC Palettes
- FRC Example Finder
- Project Explorer
- Clusters
- Loops
- Case Structure
- Shift Registers
- Enums
- State Machines

ni.com/first



FRC Vision and Robotics Palettes

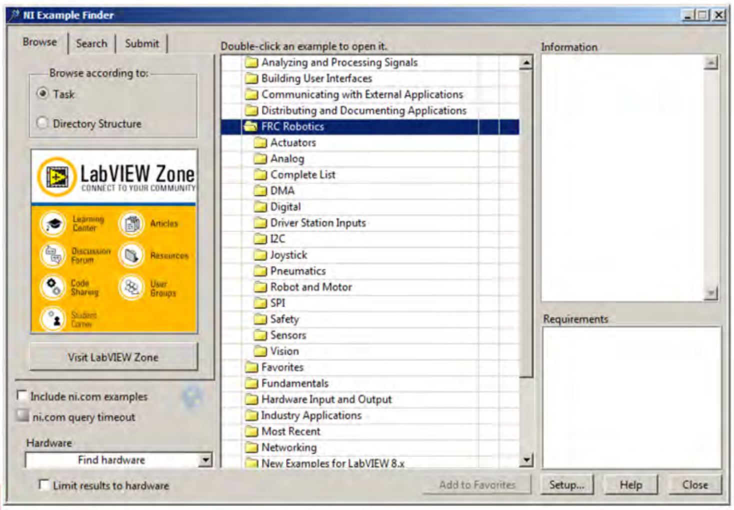
- **Vision** – manage and analyze images
- **WPI Robotics** – Robot Drive, Sensors, Actuators, I/O, Driver Station, Camera, Communications, Utilities. This contains most of what you will use to program your robot.

ni.com/first

NATIONAL INSTRUMENTS ni.com

From the LabVIEW block diagram within the Functions palette are a series of sub-palettes related to FIRST. The two primary palettes used are FIRST Vision and WPI Robotics.

Customized Example Finder



NI Example Finder

Double-click an example to open it.

- Analyzing and Processing Signals
- Building User Interfaces
- Communicating with External Applications
- Distributing and Documenting Applications
- FRC Robotics
  - Actuators
  - Analog
  - Complete List
  - DMA
  - Digital
  - Driver Station Inputs
  - I2C
  - Joystick
  - Pneumatics
  - Robot and Motor
  - SPI
  - Safety
  - Sensors
  - Vision
- Favorites
- Fundamentals
- Hardware Input and Output
- Industry Applications
- Most Recent
- Networking
- New Examples for LabVIEW 8.x

Browse according to:

● Task

○ Directory Structure

□ Include ni.com examples

□ ni.com query timeout

Hardware: Find hardware

□ Limit results to hardware

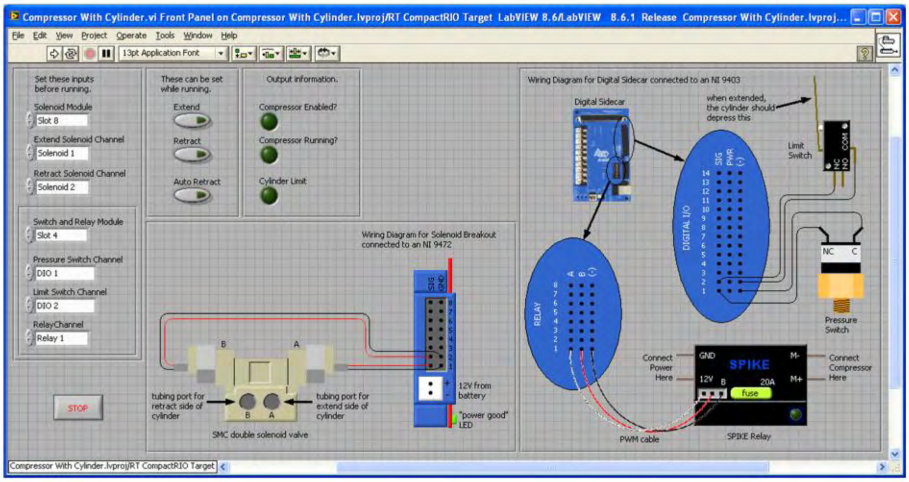
FRC

ni.com/first

NATIONAL INSTRUMENTS ni.com

Accessible from the Getting Started launch window of LabVIEW FRC is an option to “Find FRC Examples”. Here you will find may different examples of LabVIEW code to interface with different components of the robot.

FRC Specific Examples



Compressor With Cylinder.vi Front Panel on Compressor With Cylinder.lvproj/RT CompactRIO Target LabVIEW 8.6/LabVIEW 8.6.1 Release Compressor With Cylinder.lvproj...

Set these inputs before running.

Solenoid Module: Slot 8

Extend Solenoid Channel: Solenoid 1

Retract Solenoid Channel: Solenoid 2

Switch and Relay Module: Slot 4

Pressure Switch Channel: DIO 1

Limit Switch Channel: DIO 2

RelayChannel: Relay 1

These can be set while running.

Output information.

Wiring Diagram for Solenoid Breakout connected to an NI 9472

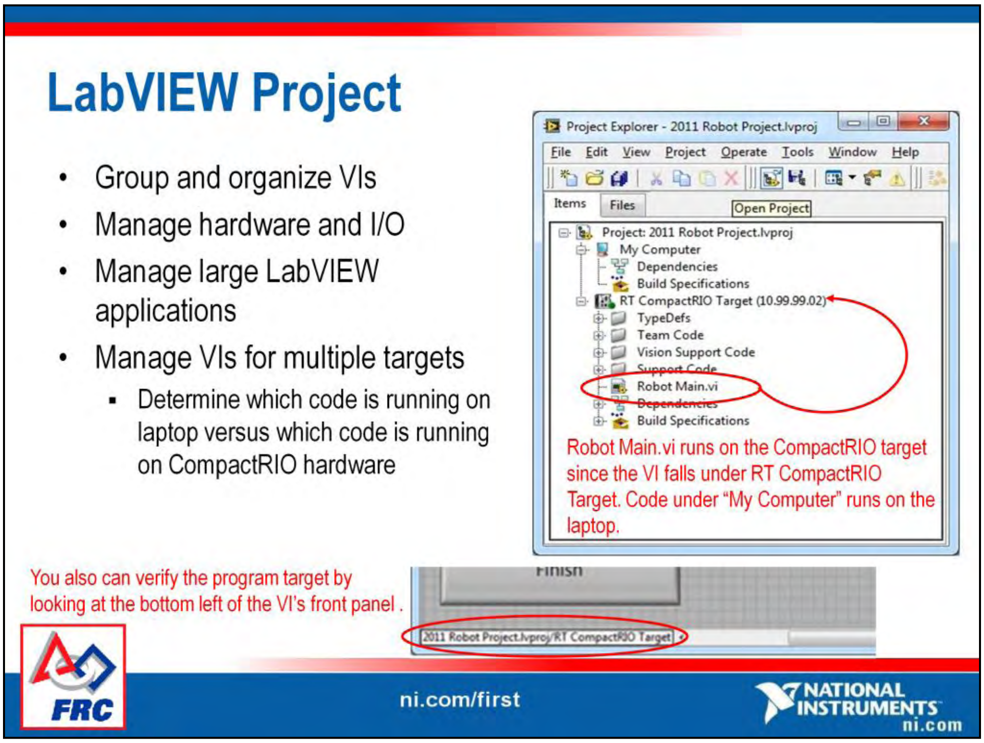
Wiring Diagram for Digital Sidecar connected to an NI 9403

Compressor With Cylinder.lvproj/RT CompactRIO Target

ni.com/first

From the Find FRC Examples:

- There are 44 FRC specific examples – more added every year
- They tell you what to configure before running, and what can be changed while running.
- They all include wiring diagrams to aid in setting up and troubleshooting your electrical system – even if you don’t program in LabVIEW!



LabVIEW Project

- Group and organize VIs
- Manage hardware and I/O
- Manage large LabVIEW applications
- Manage VIs for multiple targets
  - Determine which code is running on laptop versus which code is running on CompactRIO hardware

Project Explorer - 2011 Robot Project.lvproj
File Edit View Project Operate Tools Window Help
Items
Files
Open Project
Project: 2011 Robot Project.lvproj
My Computer
Dependencies
Build Specifications
RT CompactRIO Target (10.99.99.02)
TypeDefs
Team Code
Vision Support Code
Support Code
Robot Main.vi
Dependencies
Build Specifications
Finish
2011 Robot Project.lvproj/RT CompactRIO Target
FRC
ni.com/first
NATIONAL INSTRUMENTS
ni.com

Robot Main.vi runs on the CompactRIO target since the VI falls under RT CompactRIO Target. Code under "My Computer" runs on the laptop.

You also can verify the program target by looking at the bottom left of the VI's front panel .

LabVIEW Project

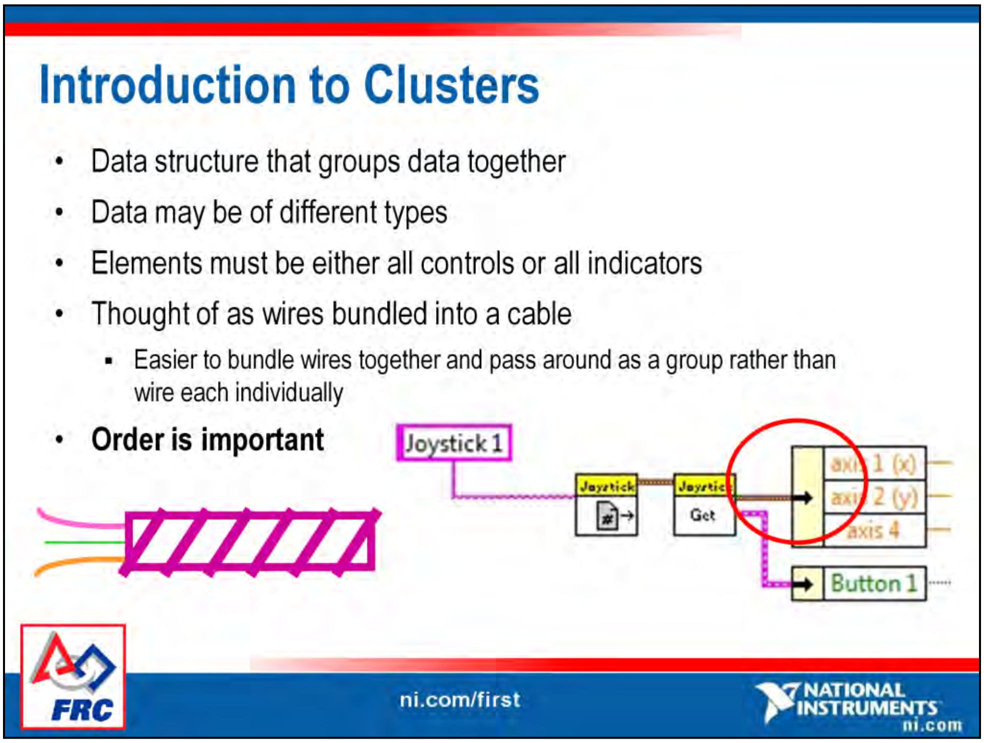
Use projects to group together LabVIEW and other files, create build specifications, and deploy or download files to targets. A target is a device or machine on which a VI runs. When you save a project, LabVIEW creates a project file (.lvproj), which includes configuration information, build information, deployment information, and references to files in the project.

You must use a project to build stand-alone applications and shared libraries. You also must use a project to work with a real-time, FPGA, or PDA target. Refer to the specific module documentation for more information about using projects with the LabVIEW Real-Time, LabVIEW FPGA, and LabVIEW Mobile modules.

Project-style LabVIEW Plug and Play instrument drivers use the project and project library features in LabVIEW 8.6. You can use project-style drivers the same you used previous LabVIEW Plug and Play drivers.

Project Explorer Window

Use the **Project Explorer Window** to create and edit projects. Select **File»New Project** to display the **Project Explorer Window**. You also can select **Project»New Project** or select **File»New** and then select **Empty Project** in the new dialog box to display the **Project Explorer Window**.



Introduction to Clusters

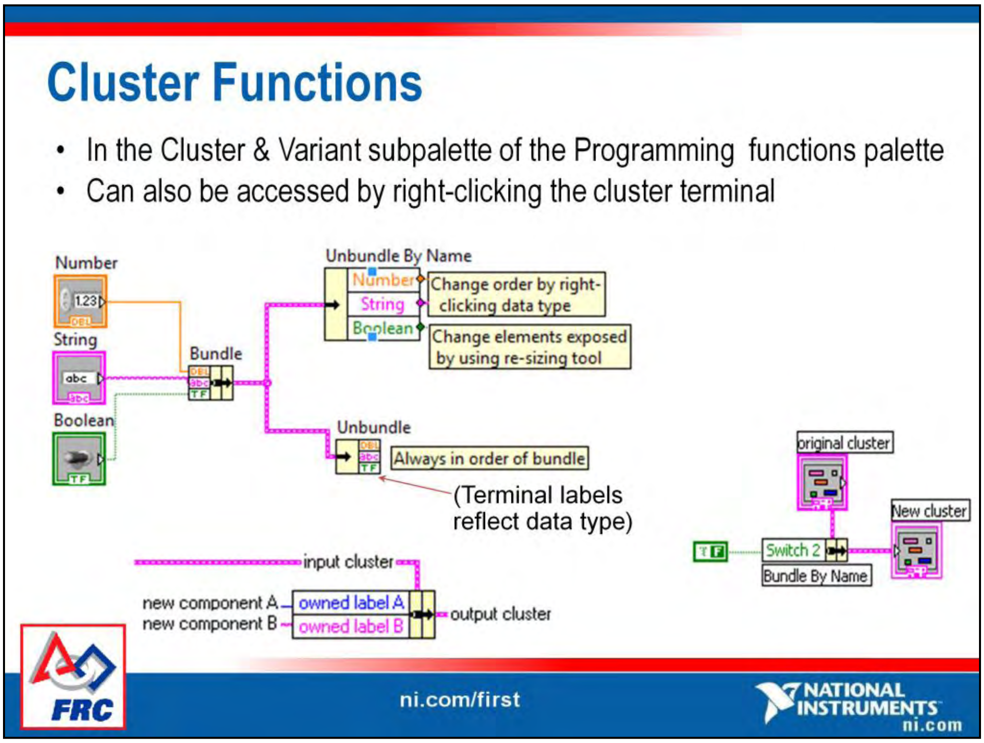
- Data structure that groups data together
- Data may be of different types
- Elements must be either all controls or all indicators
- Thought of as wires bundled into a cable
  - Easier to bundle wires together and pass around as a group rather than wire each individually
- **Order is important**

Clusters group like or unlike components together. They are equivalent to a *record* in Pascal or a *struct* in ANSI C.
Cluster components may be of different data types.

Examples

- Error information—Grouping a Boolean error flag, a numeric error code, and an error source string to specify the exact error.
- User information—Grouping a string indicating a user's name and an ID number specifying the user's security code.

All elements of a cluster must be either controls or indicators. You cannot have a string control and a Boolean indicator. Think of clusters as grouping individual wires (data objects) together into a cable (cluster).



Cluster Functions

- In the Cluster & Variant subpalette of the Programming functions palette
- Can also be accessed by right-clicking the cluster terminal

Change order by right-clicking data type

Change elements exposed by using re-sizing tool

Always in order of bundle

(Terminal labels reflect data type)

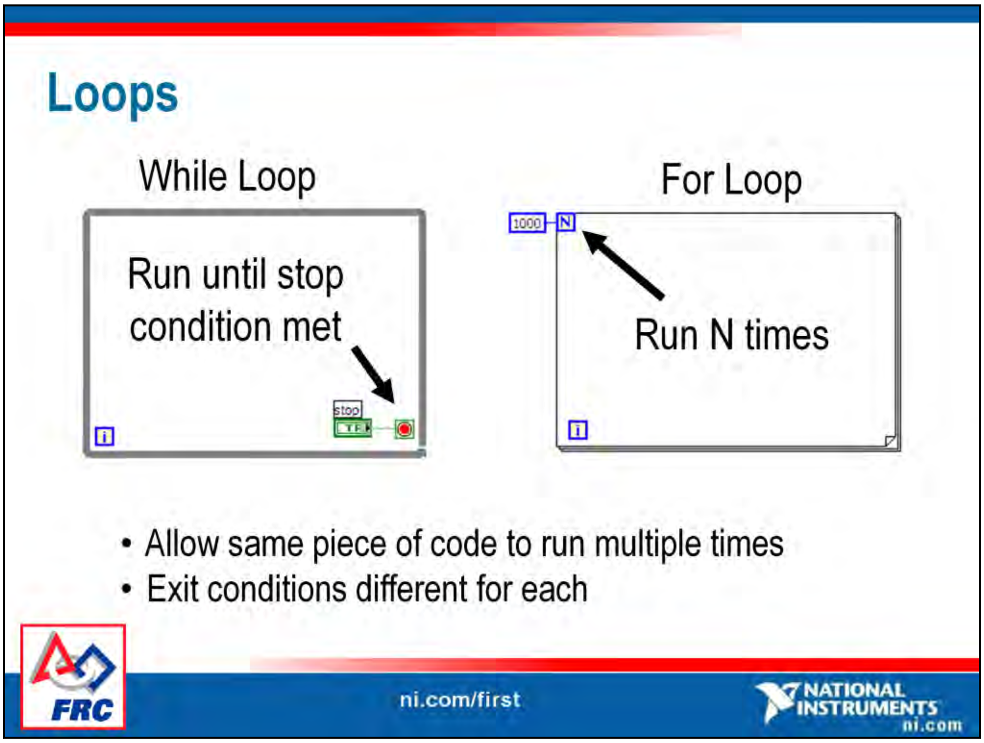
ni.com/first

The terms bundle and cluster are closely related in LabVIEW.

Example: You use a bundle function to create a cluster. You use an unbundle function to extract the parts of a cluster.

Bundle—Forms a cluster containing the given objects in the specified order.
Bundle by Name—Updates specific cluster object values (the object must have an owned label).
Unbundle—Splits a cluster into each of its individual elements by data type.
Unbundle by Name—Returns the cluster elements whose names you specify.

Note: You must have an existing cluster wired into the middle terminal of the function to use Bundle by Name.



Loops

While Loop

Run until stop condition met

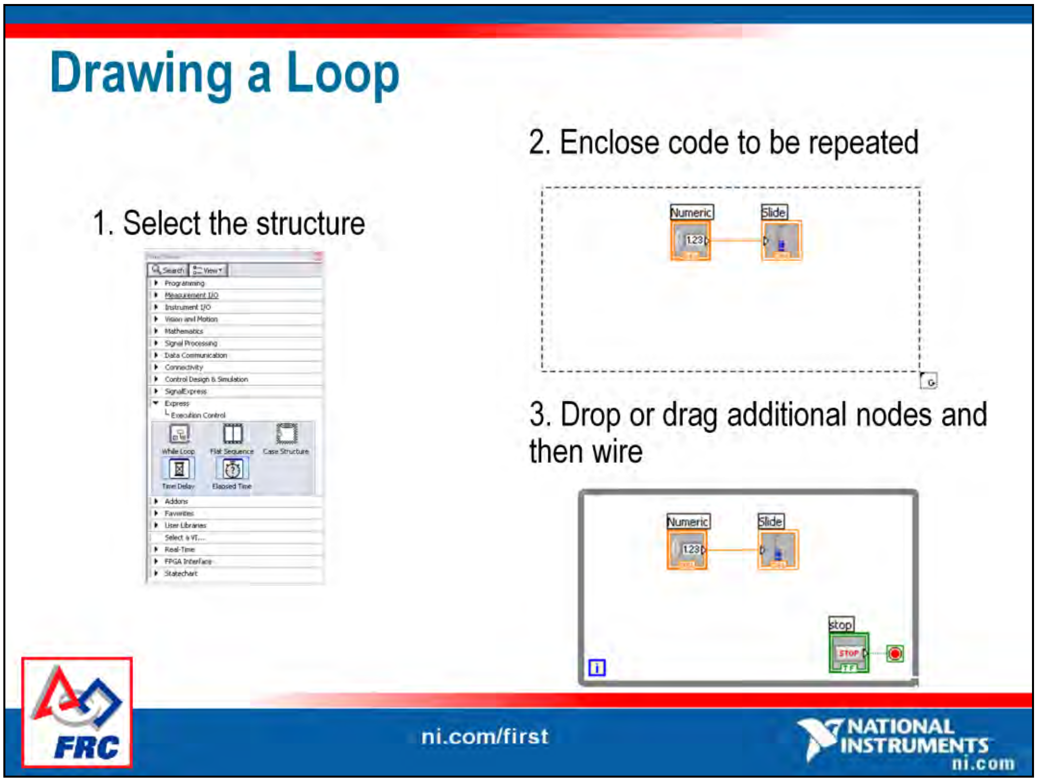
For Loop

Run N times

- Allow same piece of code to run multiple times
- Exit conditions different for each

Use While or For Loops to enable sections of your LabVIEW code to run repeatedly. A While Loop will continue to execute until a stop condition is specified. The stop condition can be a simple button press or a series of specific logical conditions. The For Loop will execute a predetermined number of times as specified by the number of iterations you wire to the N input. You may also connect an array wire to the edge of a For Loop and leave the N input unwired. The For Loop’s number of iterations will be determined by the array size that is wired at its edge. This is called auto-indexing.

To find the While and For Loops, as well as other control structures, left-click on any empty space on the block diagram and navigate to **Programming»Structures**.



Drawing a Loop

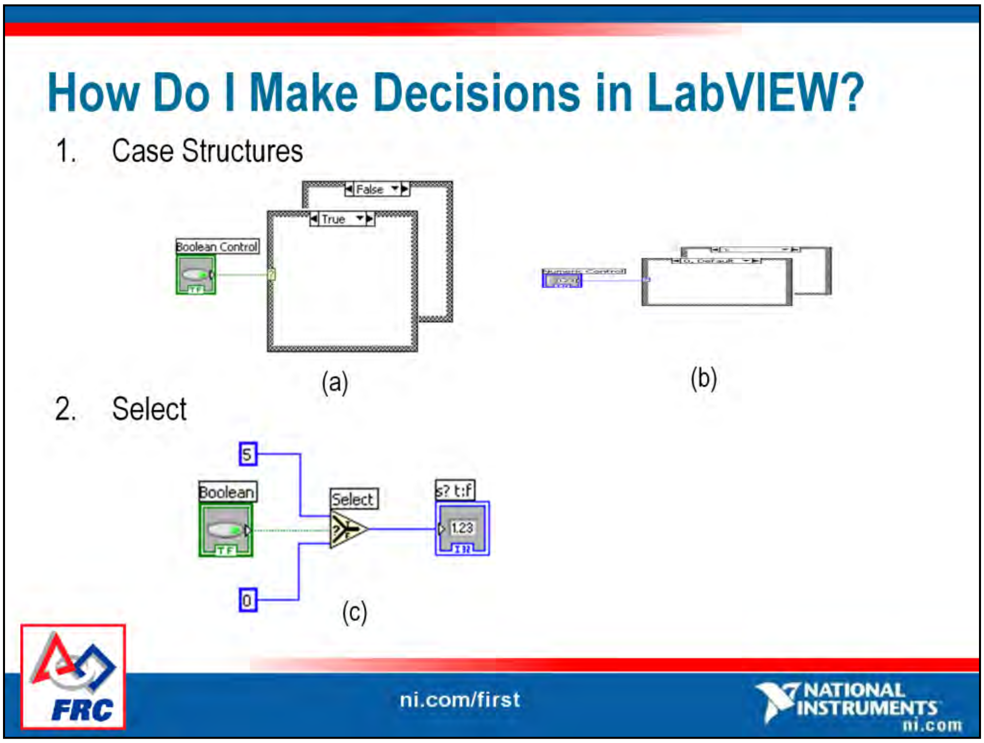
1. Select the structure

2. Enclose code to be repeated

3. Drop or drag additional nodes and then wire

Place loops in your diagram by selecting them from the **Functions** palette:

- When selected, the mouse cursor becomes a special pointer that you use to enclose the section of code you want to include in the While Loop.
- Click the mouse button to define the top-left corner and then click the mouse button again at the bottom-right corner. The While Loop boundary appears around the selected code.
- Drag or drop additional nodes in the While Loop if needed.



How Do I Make Decisions in LabVIEW?

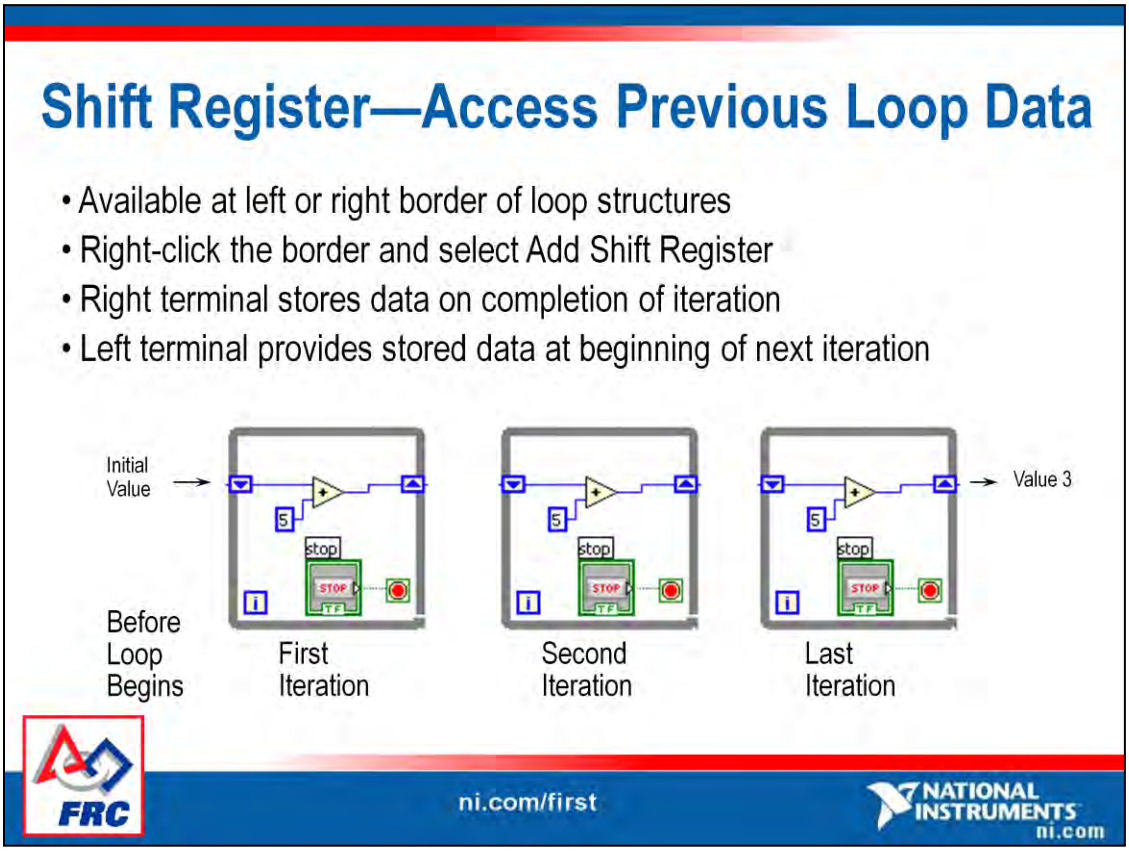
1. Case Structures

2. Select

Case Structure
The case structure has one or more subdiagrams, or cases, one of which executes when the structure executes. The value wired to the selector terminal determines which case to execute and can be Boolean, string, integer, or enumerated. Right-click the structure border to add or delete cases. Use the Labeling tool to enter value(s) in the case selector label and configure the value(s) handled by each case. It is found at **Functions»Programming»Structures»Case Structure**.

Select
Returns the value wired to the **t** input or **f** input, depending on the value of **s**. If **s** is TRUE, this function returns the value wired to **t**. If **s** is FALSE, this function returns the value wired to **f**. The connector pane displays the default data types for this polymorphic function. It is found at **Functions»Programming»Comparison»Select**.

- **Example A:** Boolean input—Simple if-then case. If the Boolean input is TRUE, the true case executes; otherwise the FALSE case executes.
- **Example B:** Numeric input. The input value determines which box to execute. If out of range of the cases, LabVIEW chooses the default case.
- **Example C:** When the Boolean passes a TRUE value to the Select VI, the value 5 is passed to the indicator. When the Boolean passes a FALSE value to the Select VI, 0 is passed to the indicator.



Shift Register—Access Previous Loop Data

- Available at left or right border of loop structures
- Right-click the border and select Add Shift Register
- Right terminal stores data on completion of iteration
- Left terminal provides stored data at beginning of next iteration

Initial Value

Before Loop Begins

First Iteration

Second Iteration

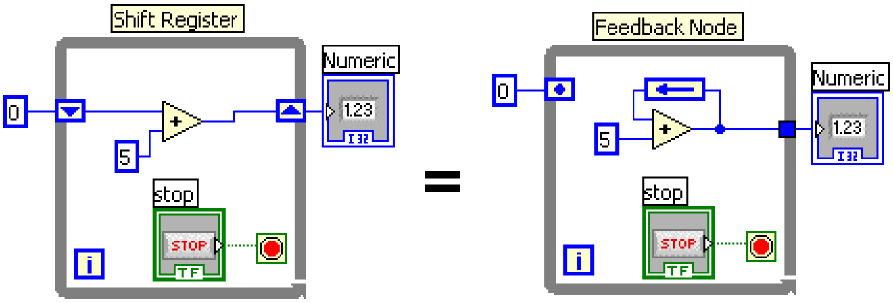
Last Iteration

Value 3

Shift registers transfer data from one iteration to the next:

- Right-click on the left or right side of a For Loop or a While Loop and select Add Shift Register.
- The right terminal stores data at the end of an iteration. Data appears at the left terminal at the start of the next iteration.
- A shift register adapts to any data type wired into it.

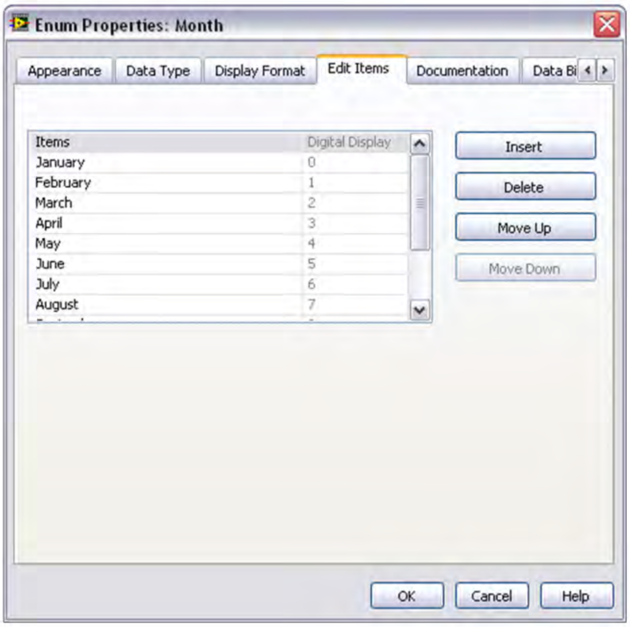
An input of 0 would result in an output of 5 on the first iteration, 10 on the second iteration, and 15 on the third iteration. Said another way, shift registers are used to retain values from one iteration to the next. They are valuable for many applications that have memory or feedback between states.



See **Help»Search the LabVIEW Help…** for more information.

Enum

An enum represents a pair of values, a string, and a numeric, where the enum can be one of a defined list of values



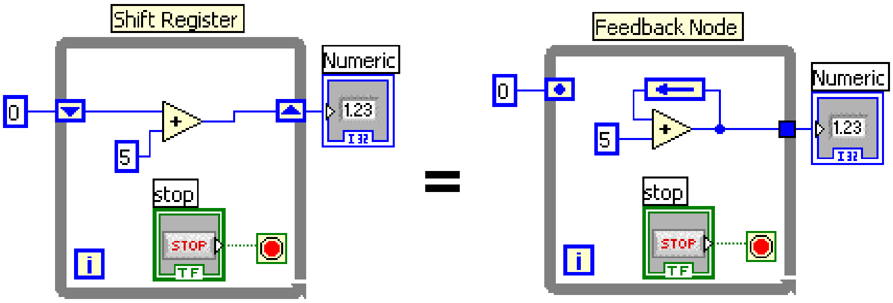
Enum Properties: Month

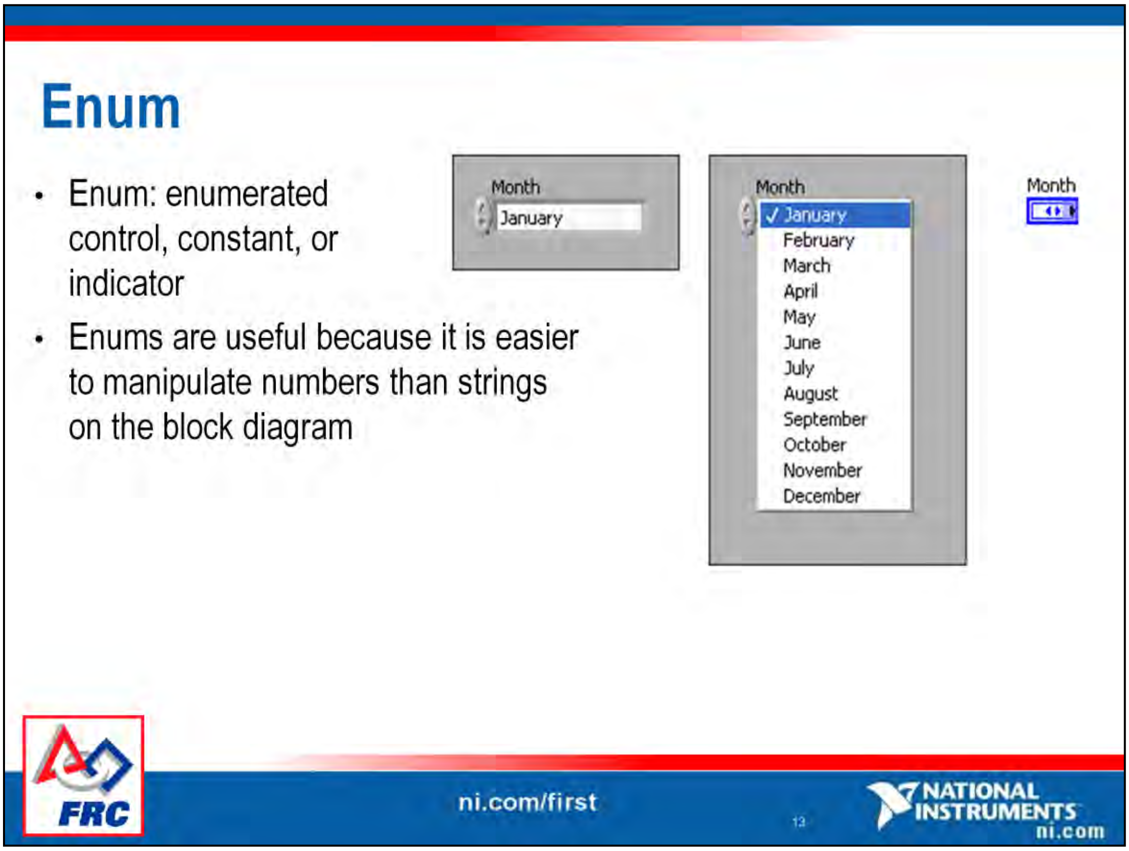
Items	Digital Display
January	0
February	1
March	2
April	3
May	4
June	5
July	6
August	7

ni.com/first

12

NATIONAL INSTRUMENTS ni.com

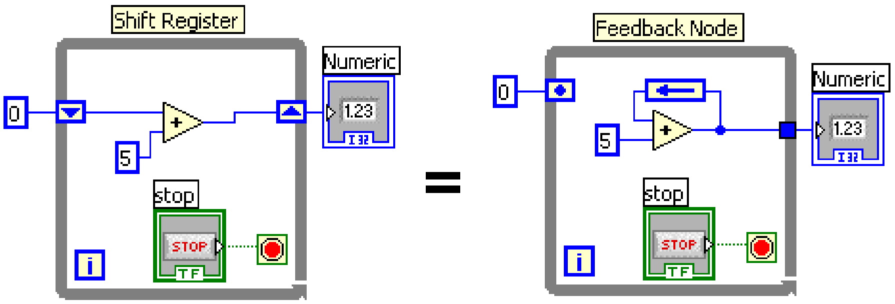


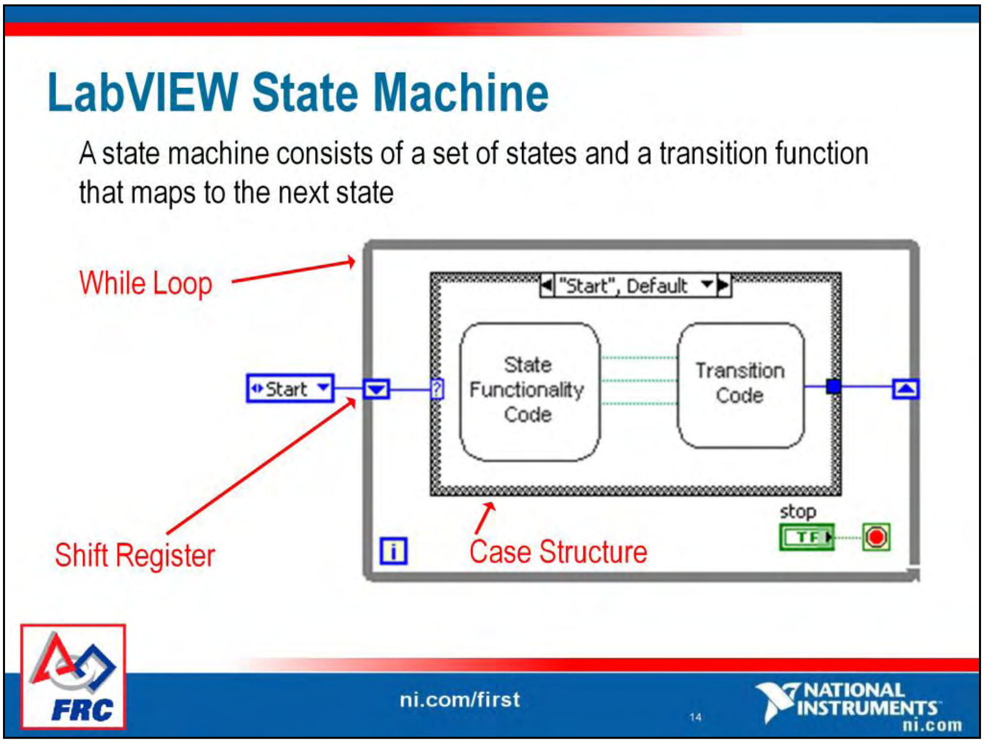


Enum

- Enum: enumerated control, constant, or indicator
- Enums are useful because it is easier to manipulate numbers than strings on the block diagram

Text appears on the front panel control, but the terminal on the block diagram sends out a numeric value which can be move easy to manipulate.





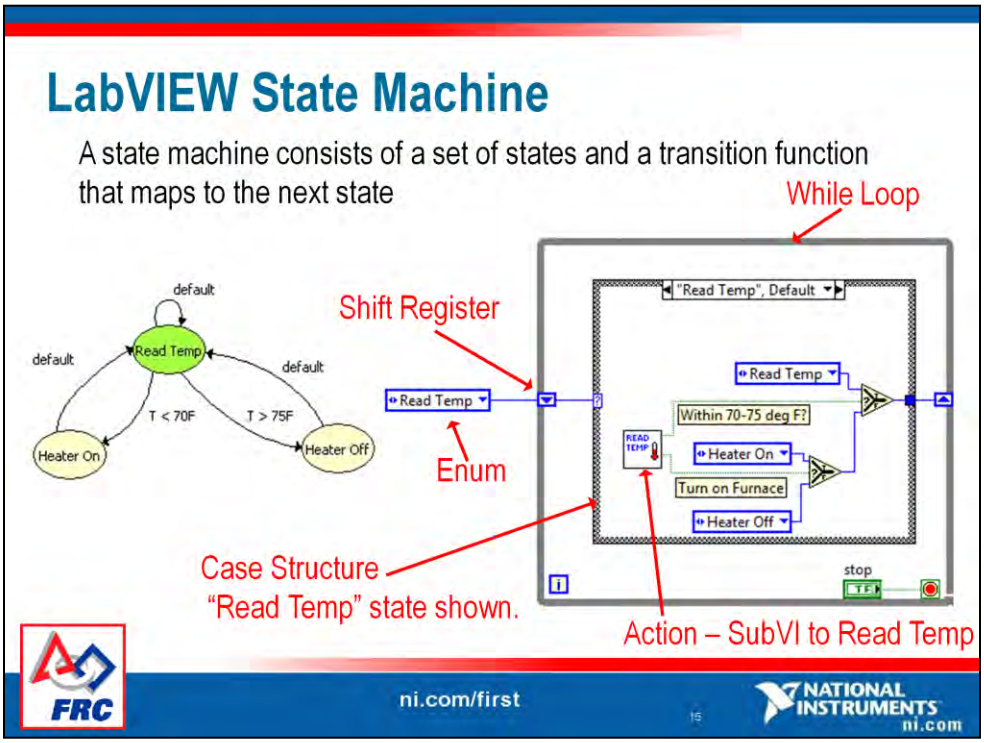
LabVIEW State Machine

A state machine consists of a set of states and a transition function that maps to the next state

ni.com/first

14

One of the most important decision-making structures in LabVIEW is the case structure. The case structure enables you to encapsulate sections of code so they run only when a specific condition that you control occurs. The case structure changes states based on its input value. For example, you might have three states defined in your program: initialize, collect data, and report data. The code for these different states would reside within the boundaries of the case structure, and you would switch between the states with either buttons on a front panel or some combination of logic built into your LabVIEW application.



LabVIEW State Machine

A state machine consists of a set of states and a transition function that maps to the next state

While Loop

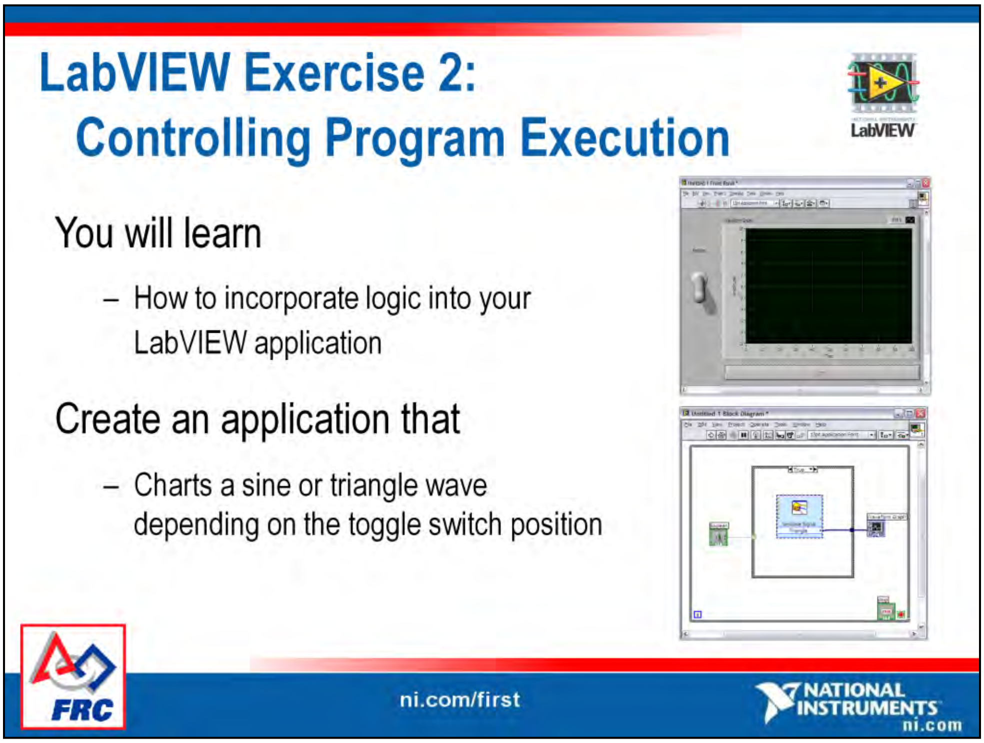
Shift Register

Enum

Case Structure
"Read Temp" state shown.

Action – SubVI to Read Temp

A real-world example of a state machine using the furnace example state diagram.



LabVIEW Exercise 2:
Controlling Program Execution

You will learn

- How to incorporate logic into your LabVIEW application

Create an application that

- Charts a sine or triangle wave depending on the toggle switch position

ni.com/first

Exercise 2: Controlling Program Execution

If you are up for a challenge, try to build the following LabVIEW application without using the instructions; if you feel you need more guidance, skip down to the step-by-step section for a complete set of instructions.

Challenge Application

Using a While Loop, case structure, toggle switch, and the Random Number Generator VI, create a simple application that charts either a random number or a constant value, depending on your toggle switch position.

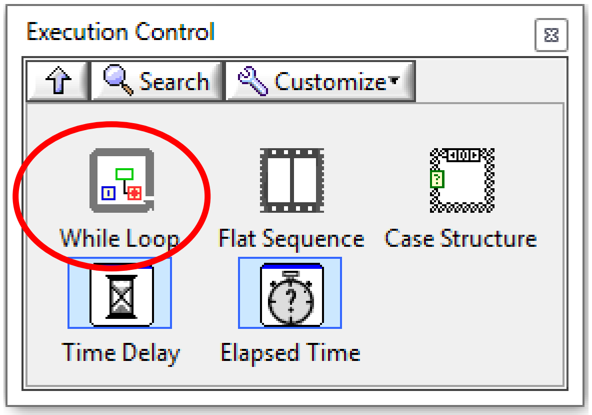
If you are really ambitious, add logic to the VI so it stops *EITHER* when you press the STOP button *OR* when the loop iterations have exceeded 1,000.

Step-by-Step Instructions

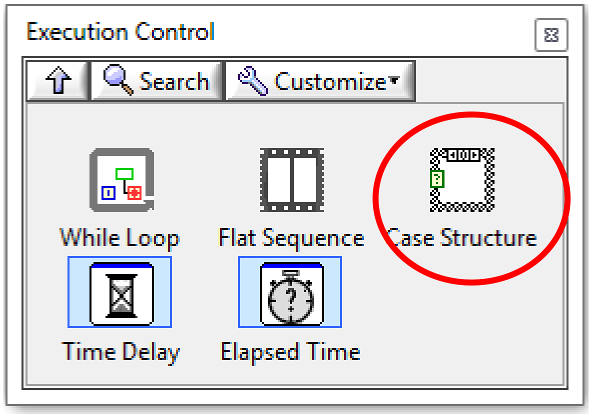
In this exercise, create a LabVIEW VI that outputs either a random number or a constant value, depending on the state of a toggle switch. Use a case structure to handle the logic of which signal is output and a While Loop to keep the application running until a Stop button is pressed.

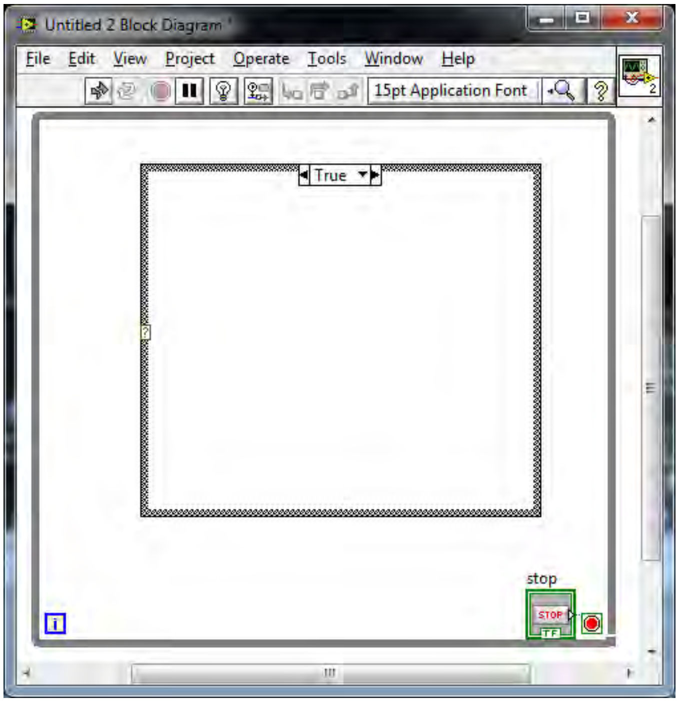
1. Create a **New Blank VI**, from the File menu.
2. Start by placing a While Loop on the block diagram. Draw it large enough to accommodate the other code you will place inside it. You can locate the While Loop on the **Express» Execution Control** palette as shown below.

   Note: Creating a while loop from the Execution Control Express Palette, auto-generated the STOP button.



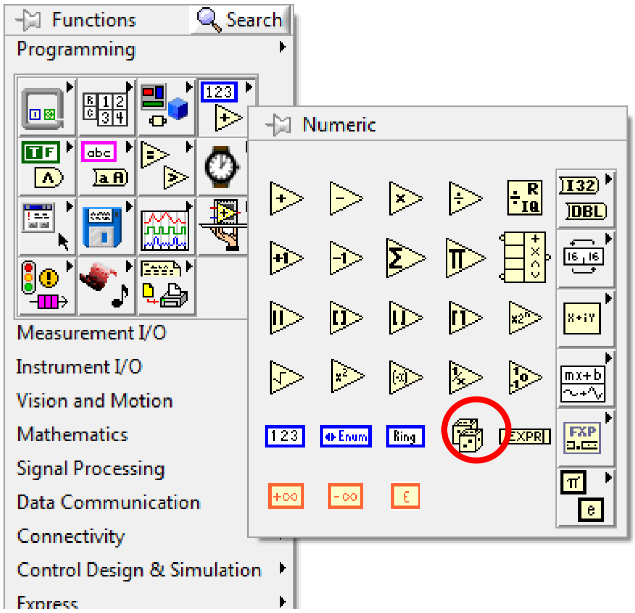
3. Next select the **Case Structure** from the same palette and place it inside the While Loop as shown below.



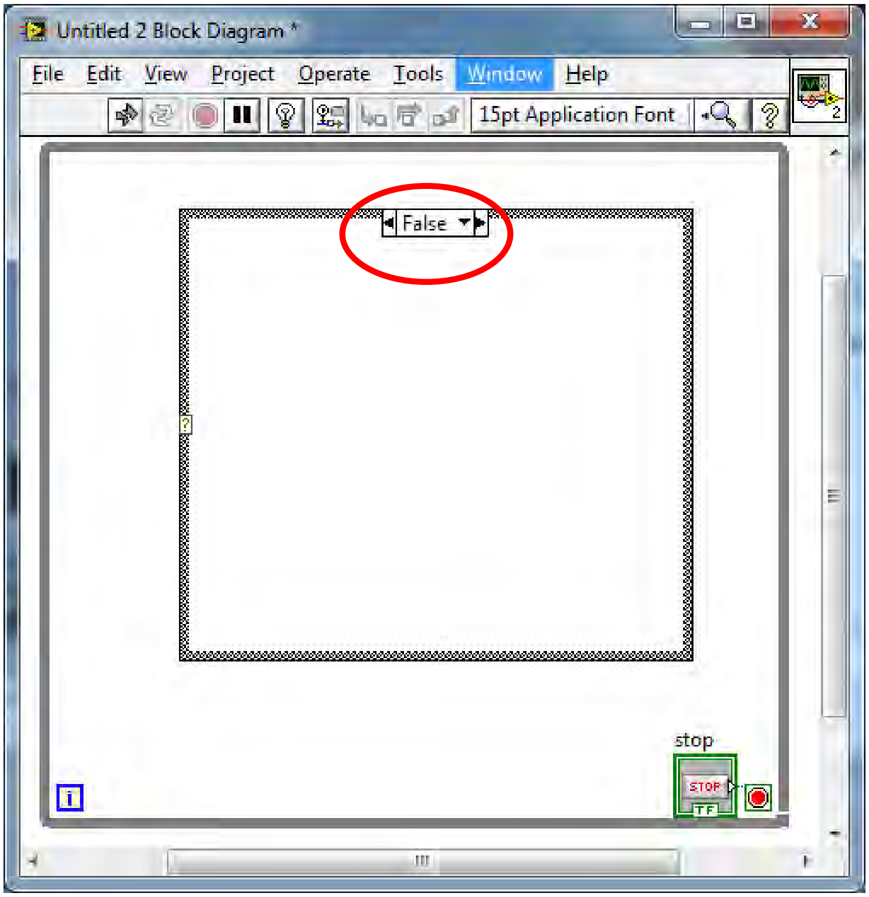


4. Then add the **Random Number Generator** to the **True** case of the case structure. On the **Functions Palette,** navigate to **Programming»Numeric**, select **Random Number (0-1)** as shown below, and place it inside the **True** case statement.

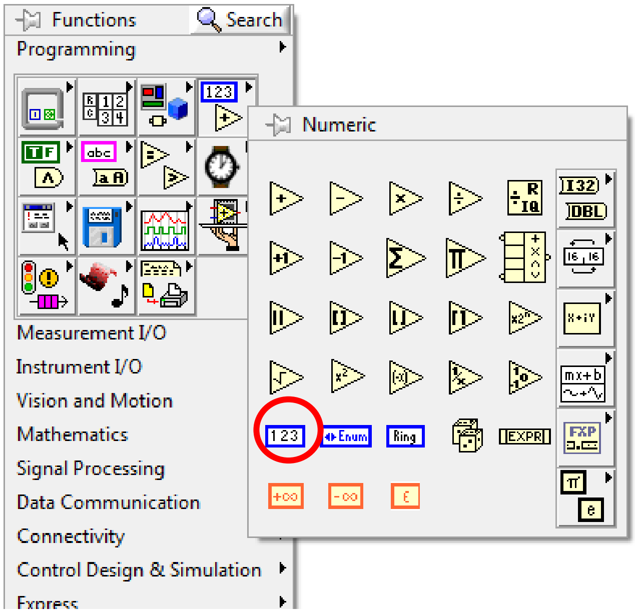
   Note that the Random Number function disappears, as it is contained inside the True case. It is still present in the Block Diagram, but the case structure only shows one case statement at a time.



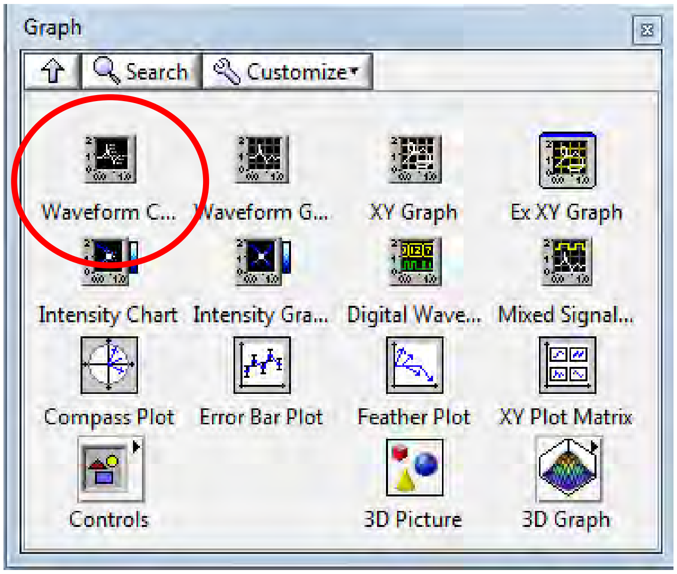
5. Next, switch the case statement from True to False, as shown below.



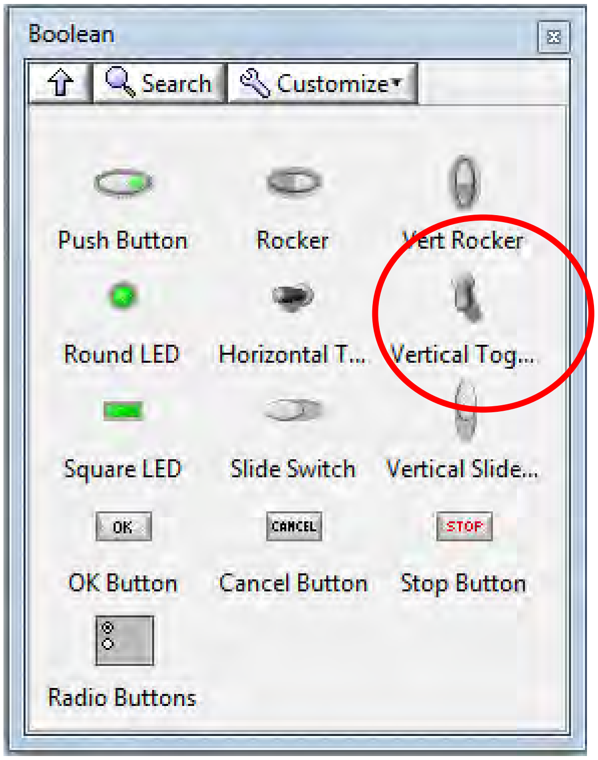
6. Now add the **Numeric Constant** to the **False** case of the case structure. On the **Functions Palette,** navigate to **Programming»Numeric**, select **Numeric Constant** as shown below, and place it inside the **False** case statement.



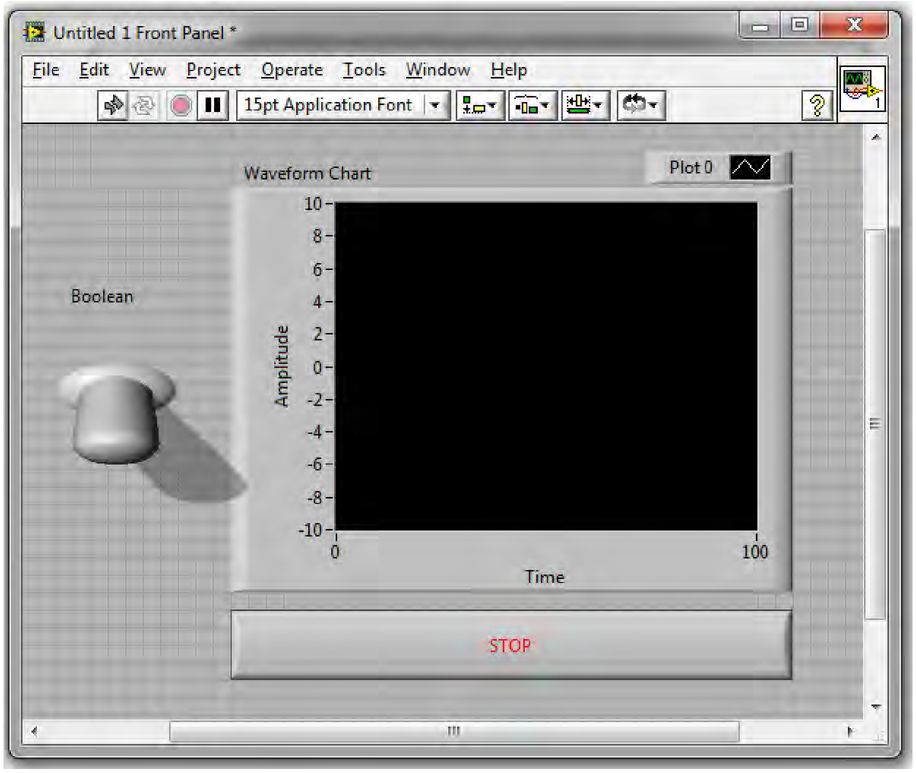
7. Switch to your front panel (Ctrl-E) and place a Waveform Chart as shown below.



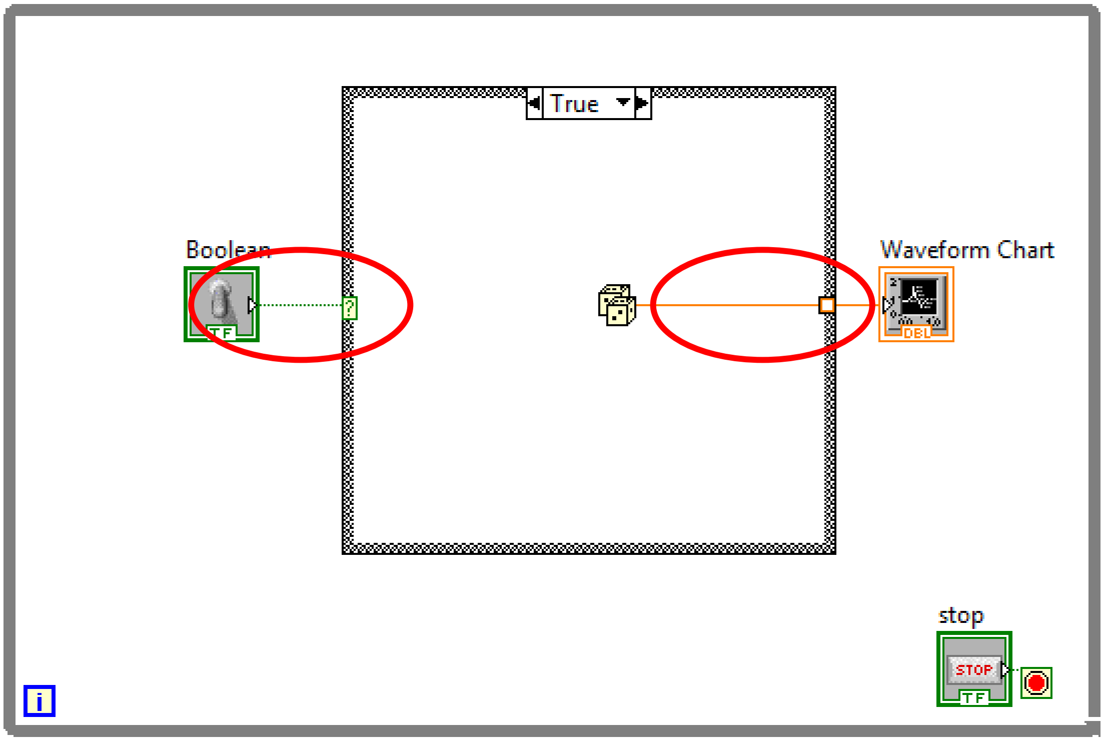
8. Also add a vertical toggle switch from the **Boolean** palette to the front panel as shown below.

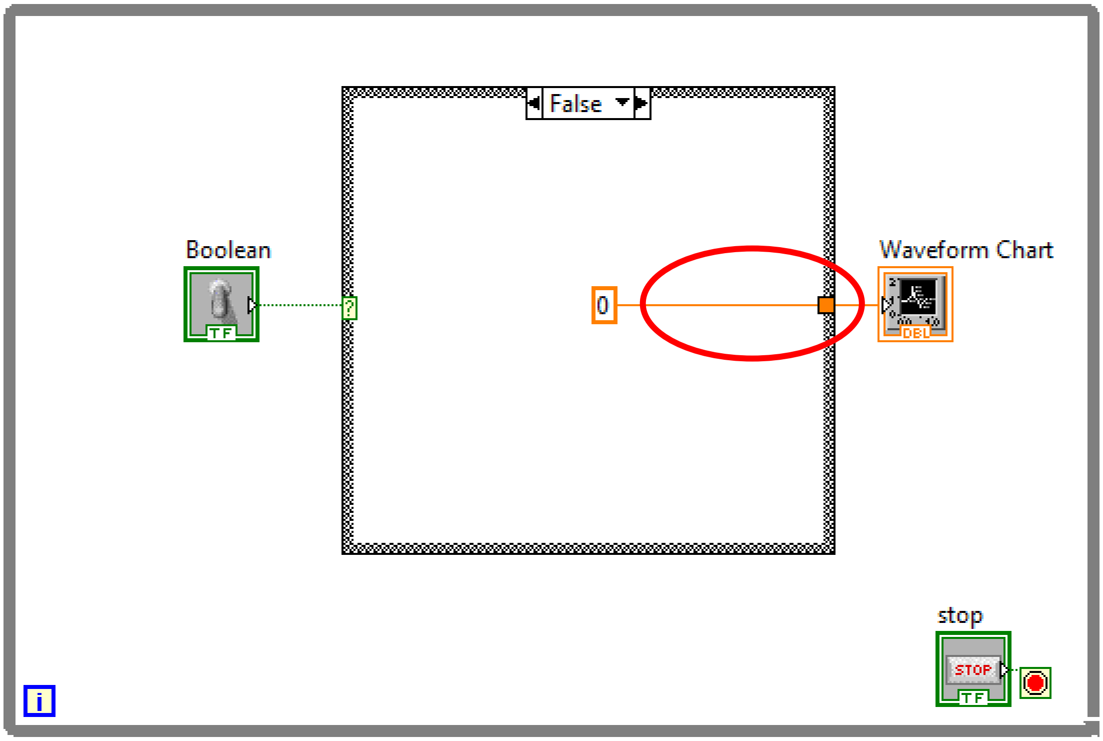


9. Your front panel should look like the following.

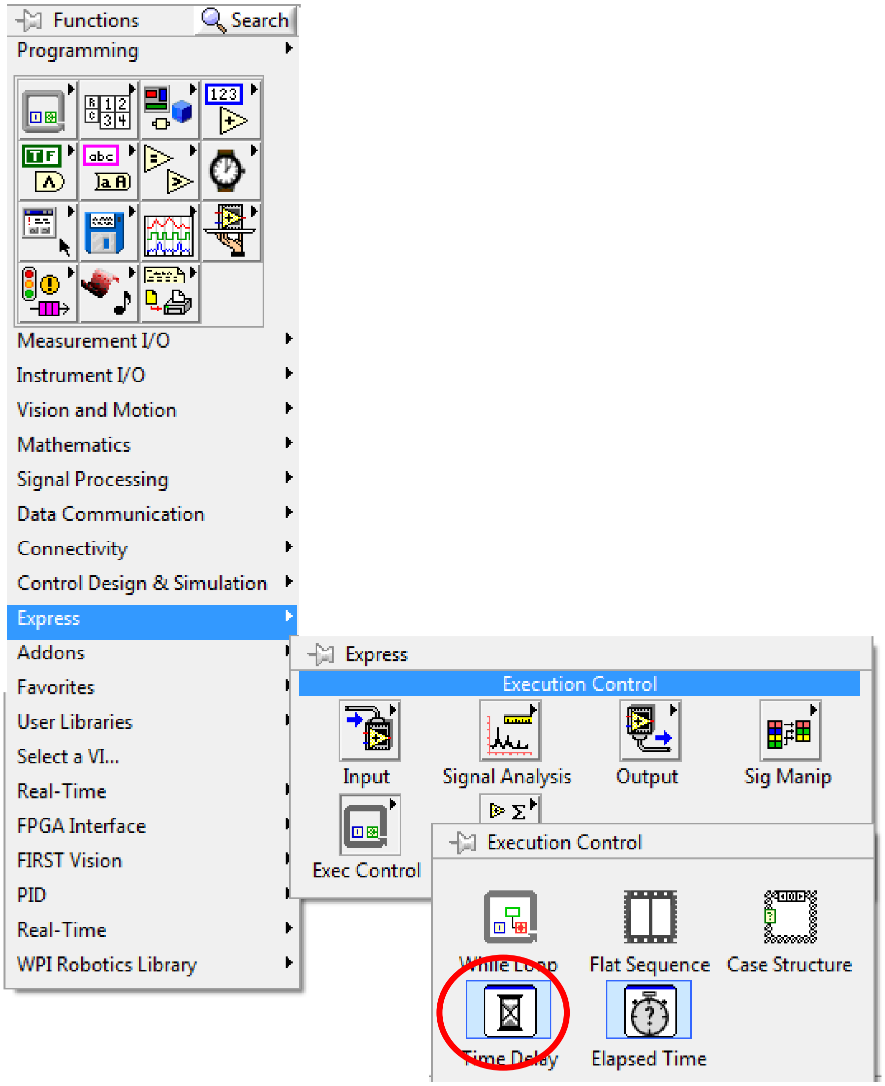


10. Switch back to the block diagram and **wire** the toggle switch to the case selector input of the case structure as shown below. Also **wire** the Numeric Constant and Random Number VI to the Waveform Chart as shown.

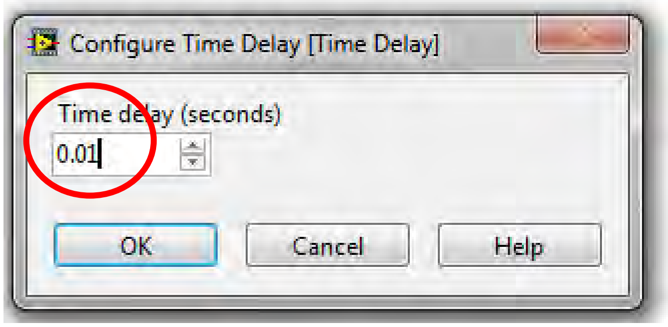




11. Next, add a time delay to slow the execution of the loop. Without this delay, the loop runs much faster than our eyes can follow and takes up a lot of the PC's processing power for no good reason. On the Functions palette, navigate to **Express»Execution Control»Time Delay** and drop the time delay within the While Loop but outside the case structure.

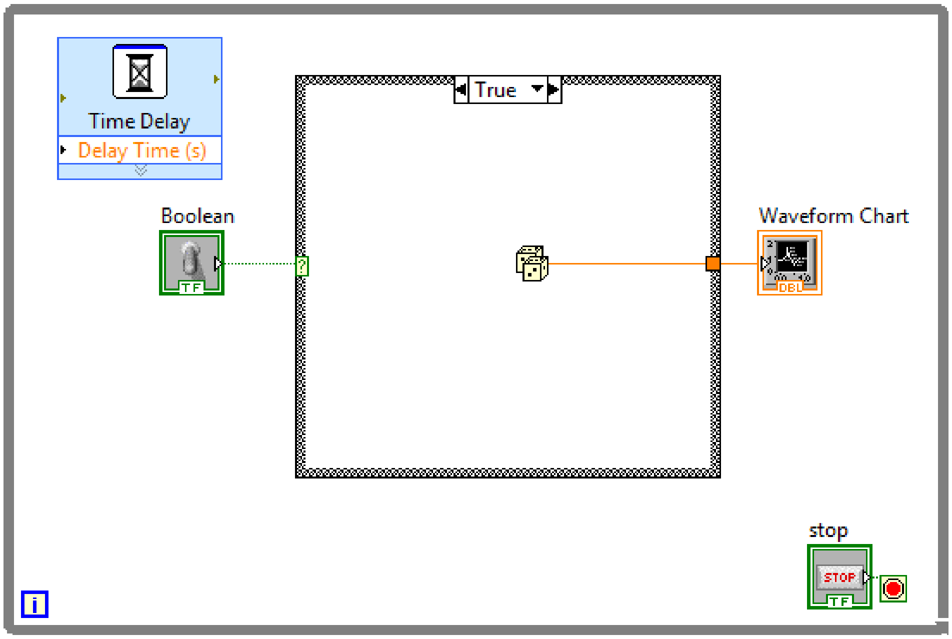


The **Time Delay** Express VI opens the following dialog box. **Set** the time delay to **0.01** s so the loop runs 100 times per second. Press **OK**.



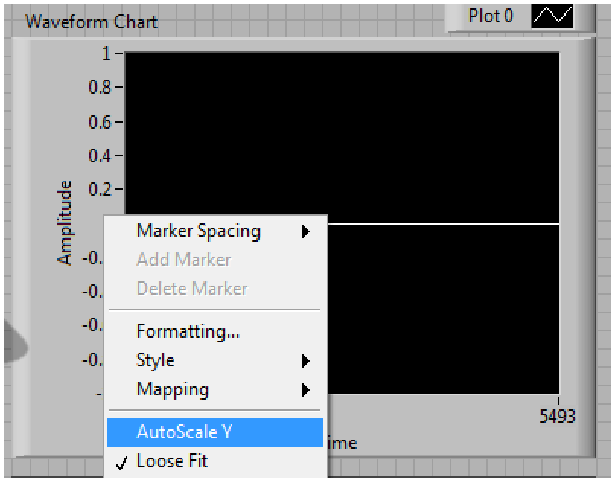
Time delay (seconds)
0.01

12. Your block diagram should resemble the one below.



13. You are ready to test your VI by running the program and switching the toggle switch back and forth. You should see the graph switch between a random and constant number on the chart.

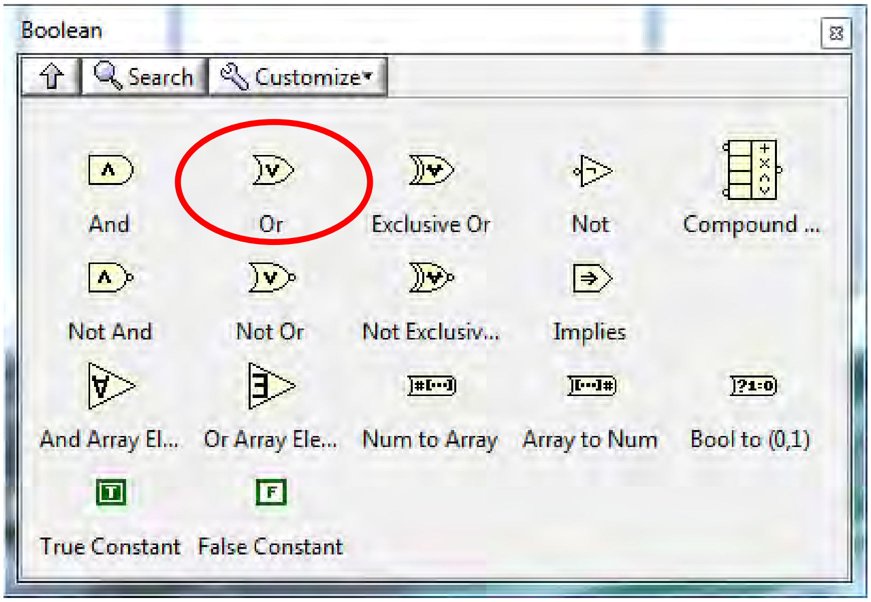
HINT: After running the program with the toggle in the "off" position, right-click the Waveform Chart from the front panel and deselect the **AutoScale Y** option. This stops the chart from autoscaling back and forth when the toggle is selected.

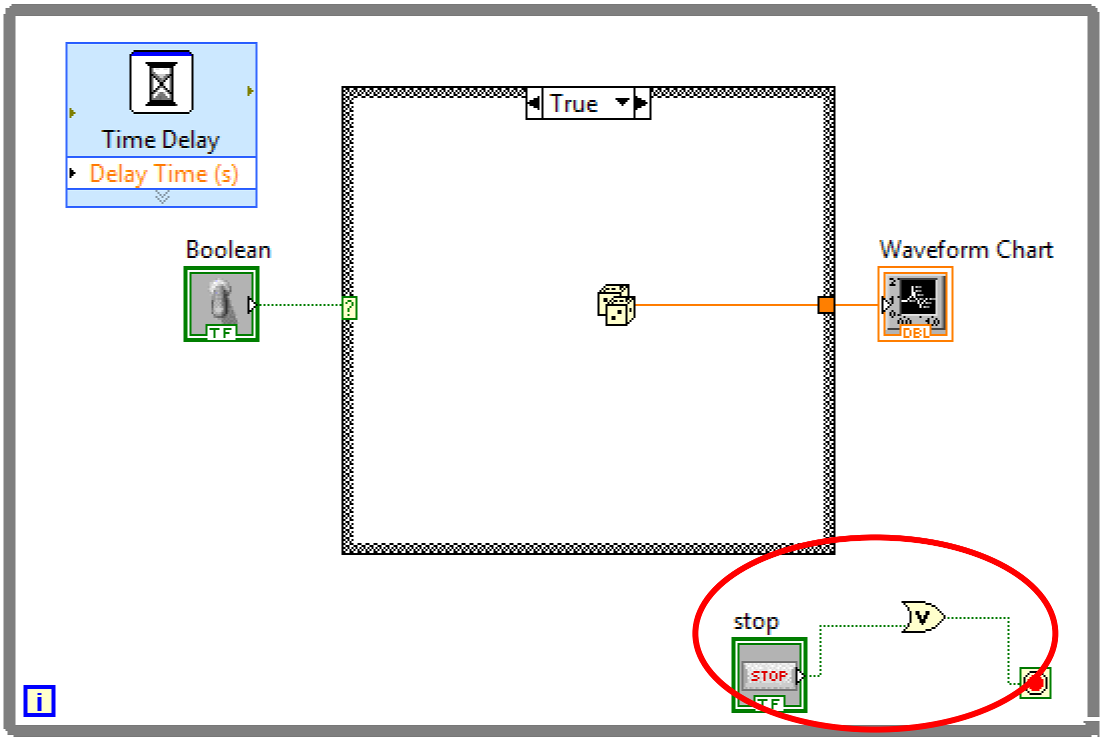


Extra Challenge

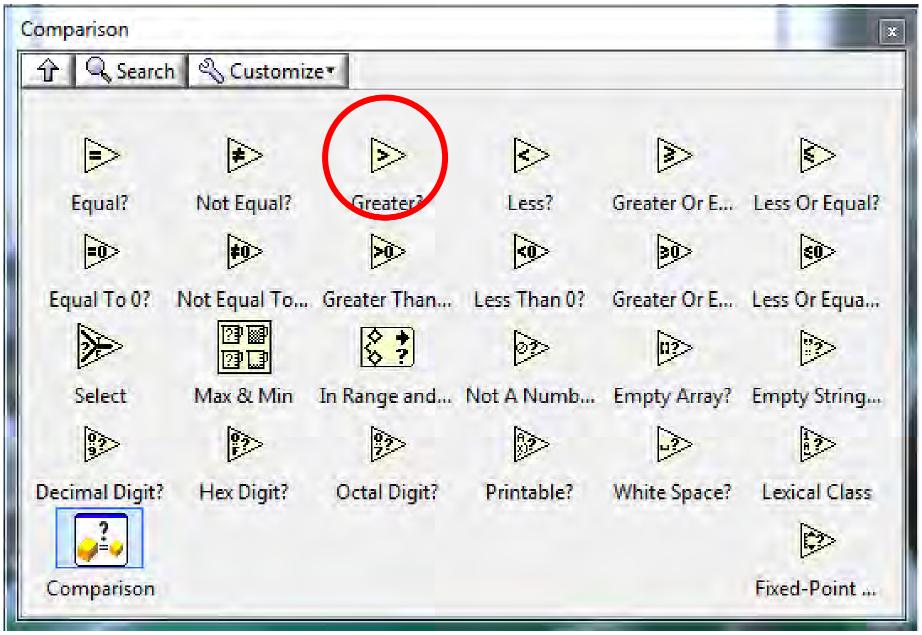
Add logic to the VI so it stops either when you press the **STOP** button or when the loop iterations have exceeded 1,000.

1. Do this by inserting an **Or** function from the **Boolean** palette before the **STOP** button, as shown below.

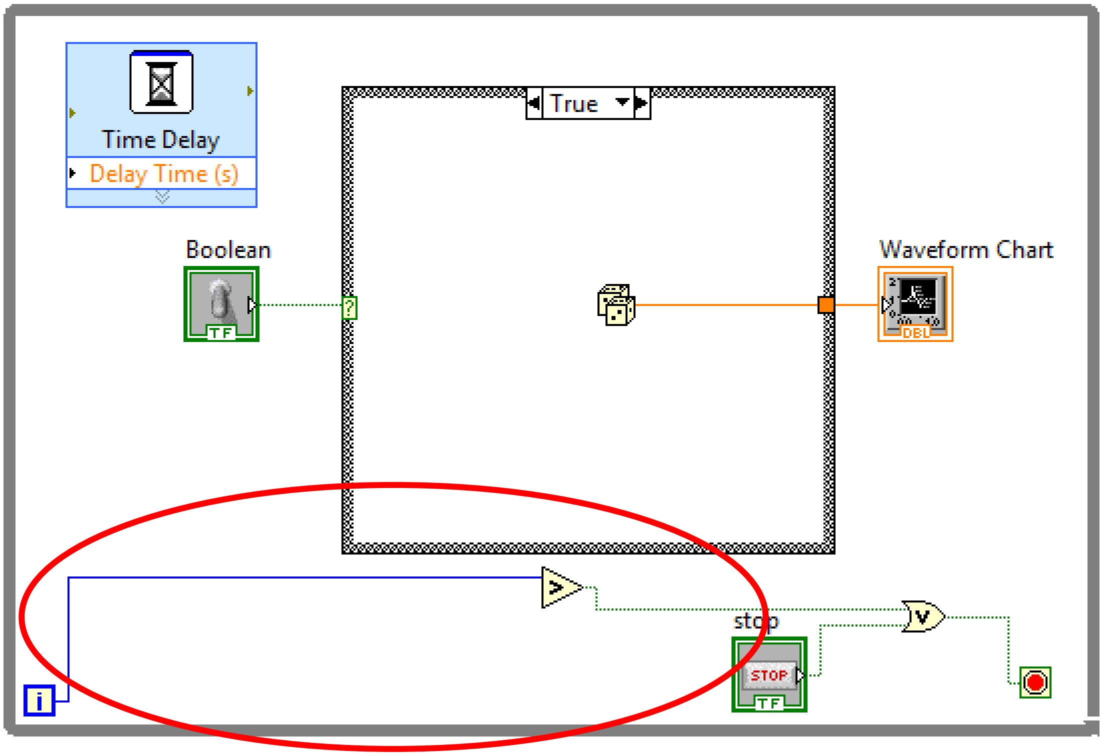




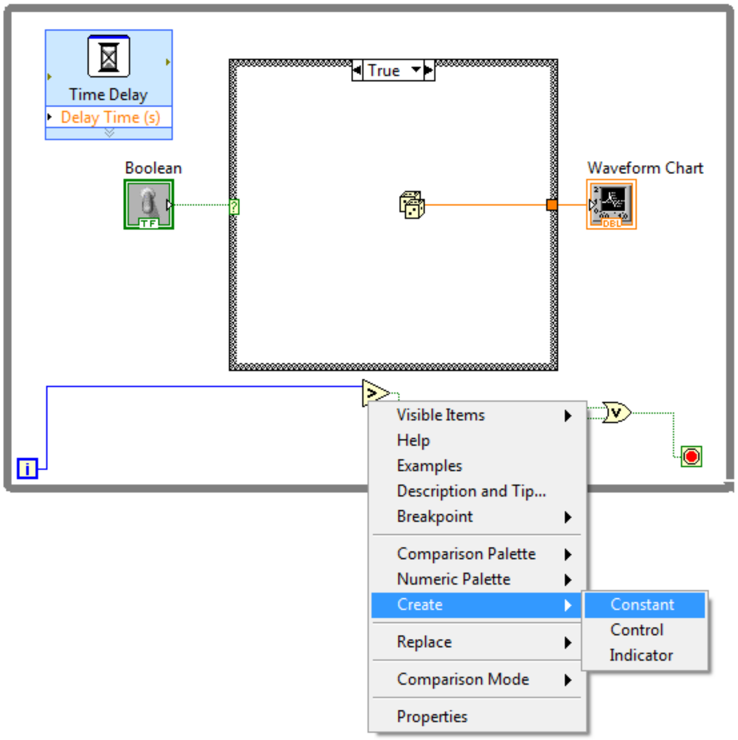
2. Next select the **Greater?** function from the Programming>>**Comparisons** palette.



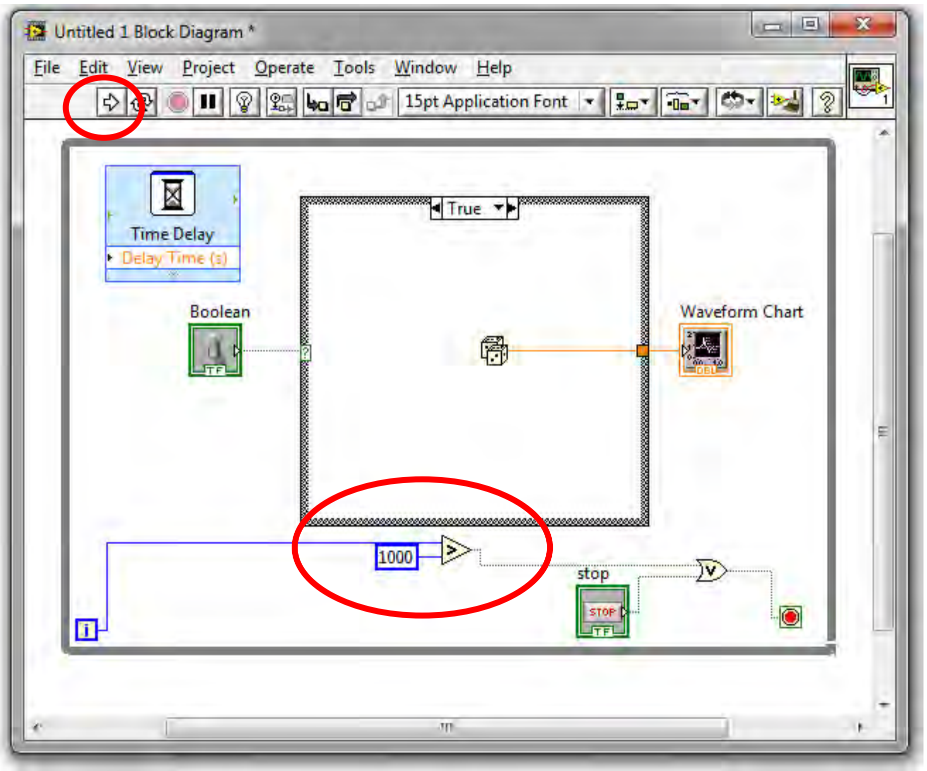
3. Wire as shown below.



4. Right-click on the open terminal of the **Greater?** block and select **Create»Constant**.



5. Change the constant to 1000 as shown below and **Run** the VI.
Given that we have defined a loop rate of 0.01s, how long do we expect the program to run?



END LABVIEW EXERCISE 2

Advanced

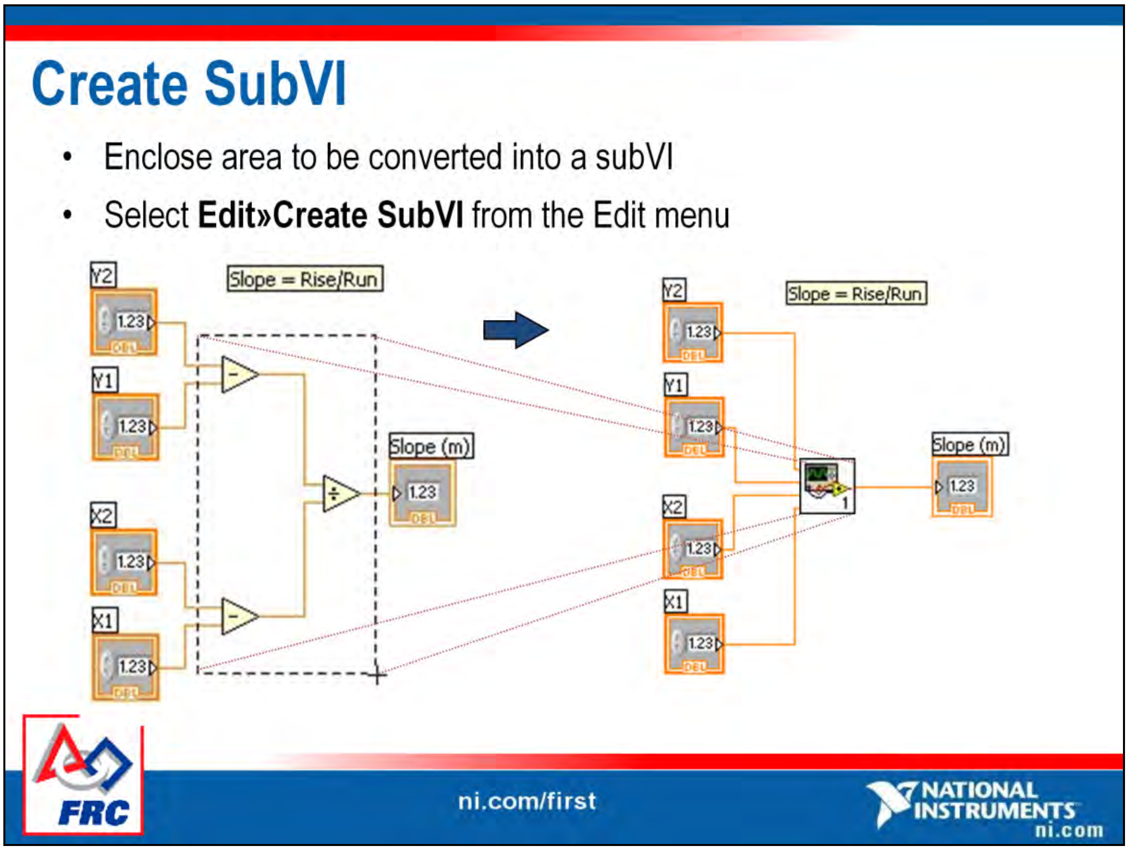
For reference but not covered in hands-on session.

FRC

ni.com/first

NATIONAL INSTRUMENTS ni.com

Useful-to-know tools, techniques, and details. Not covered in hands-on session.



Create SubVI

- Enclose area to be converted into a subVI
- Select **Edit»Create SubVI** from the Edit menu

ni.com/first

Creating SubVIs

After you build a VI, you can use it in another VI. A VI called from the block diagram of another VI is called a subVI. You can reuse a subVI in other VIs. To create a subVI, you need to build a connector pane and create an icon.

A subVI node corresponds to a subroutine call in text-based programming languages. A block diagram that contains several identical subVI nodes calls the same subVI several times.

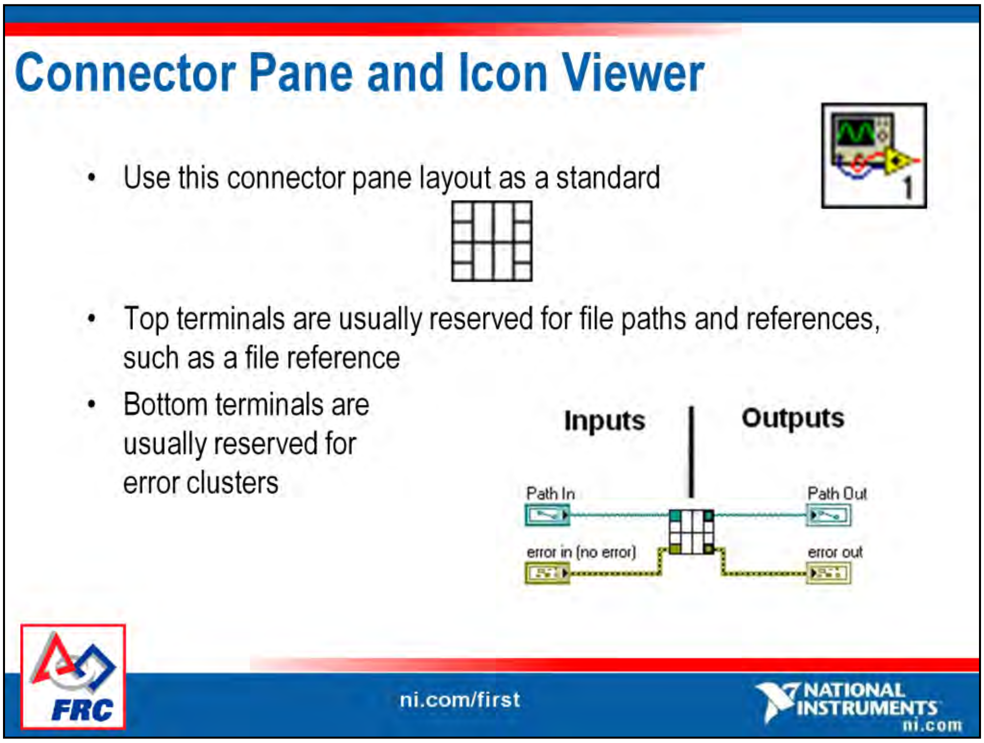
The subVI controls and indicators receive data from and return data to the block diagram of the calling VI. Click the **Select a VI** icon or text on the **Functions** palette, navigate to and double-click a VI, and place the VI on a block diagram to create a subVI call to that VI.

You can easily customize subVI input and output terminals as well as the icon. Follow the instructions below to quickly create a subVI.

Creating SubVIs From Sections of a VI

Convert a section of a VI into a subVI by using the Positioning tool to select the section of the block diagram you want to reuse and by selecting **Edit»Create SubVI**. An icon for the new subVI replaces the selected section of the block diagram. LabVIEW creates controls and indicators for the new subVI, automatically configures the connector pane based on the number of control and indicator terminals you selected, and wires the subVI to the existing wires.

See **Help»Search the LabVIEW Help…»SubVIs** for more information.



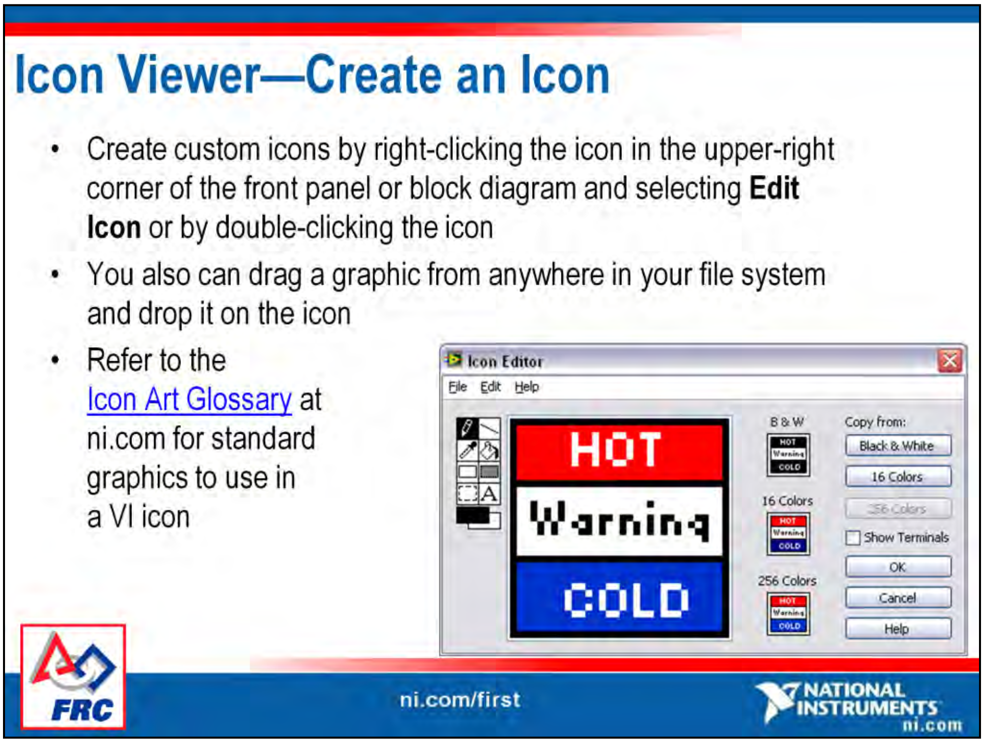
Connector Pane and Icon Viewer

- Use this connector pane layout as a standard
- Top terminals are usually reserved for file paths and references, such as a file reference
- Bottom terminals are usually reserved for error clusters

With the connector pane and icon viewer, you can define the data being transferred in and out of the subVI as well as its appearance in the main LabVIEW code. Every VI displays an icon in the upper-right corner of the front panel and block diagram windows. After you build a VI, build the connector pane and icon so you can use the VI as a subVI.

The icon and connector pane correspond to the function prototype in text-based programming languages. There are many options for the connector pane, but some general standards are specified above. Notably, always reserve the top terminals for references and the bottom terminals for error clusters.

To define a connector pane, right-click the icon in the upper-right corner of the front panel and select **Show Connector** from the shortcut menu. Each rectangle on the connector pane represents a terminal. Use the terminals to assign inputs and outputs. Select a different pattern by right-clicking the connector pane and selecting **Patterns** from the shortcut menu.



Icon Viewer—Create an Icon

- Create custom icons by right-clicking the icon in the upper-right corner of the front panel or block diagram and selecting **Edit Icon** or by double-clicking the icon
- You also can drag a graphic from anywhere in your file system and drop it on the icon
- Refer to the Icon Art Glossary at ni.com for standard graphics to use in a VI icon

An icon is a graphical representation of a VI. If you use a VI as a subVI, the icon identifies the subVI on the block diagram of the VI. The Icon Editor is a utility built into LabVIEW that you can use to fully customize the appearance of your subVIs. This allows you to visually distinguish your subVIs, which greatly improves the usability of the subVI in large portions of code.

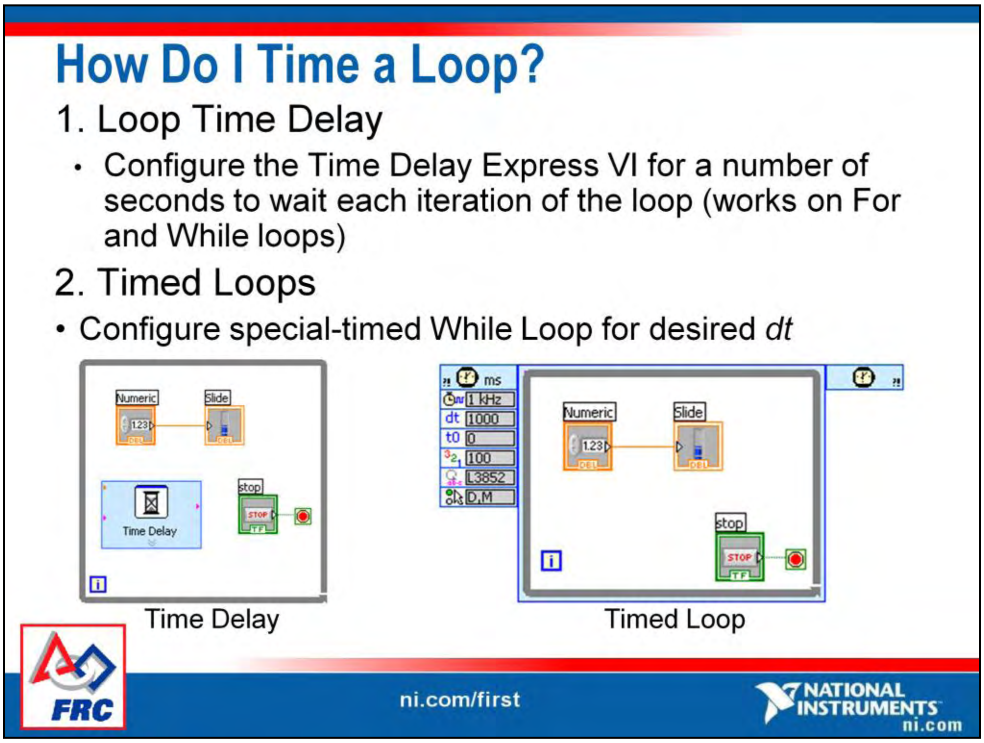
After you've defined the connector pane and have customized the icon, you are ready to place the subVI into other LabVIEW code. There are two ways to accomplish this:

To place a subVI on the block diagram

1. Select the **Select a VI…** from the **Functions** palette
2. Navigate to the VI you want to use as a subVI
3. Double-click it to place it on the block diagram

To place an open VI on the block diagram of another open VI

1. Use the **Positioning tool** to click the icon of the VI you want to use as a subVI
2. Drag the icon to the block diagram of the other VI



How Do I Time a Loop?

1. Loop Time Delay
   - Configure the Time Delay Express VI for a number of seconds to wait each iteration of the loop (works on For and While loops)
2. Timed Loops
   - Configure special-timed While Loop for desired *dt*

Time Delay

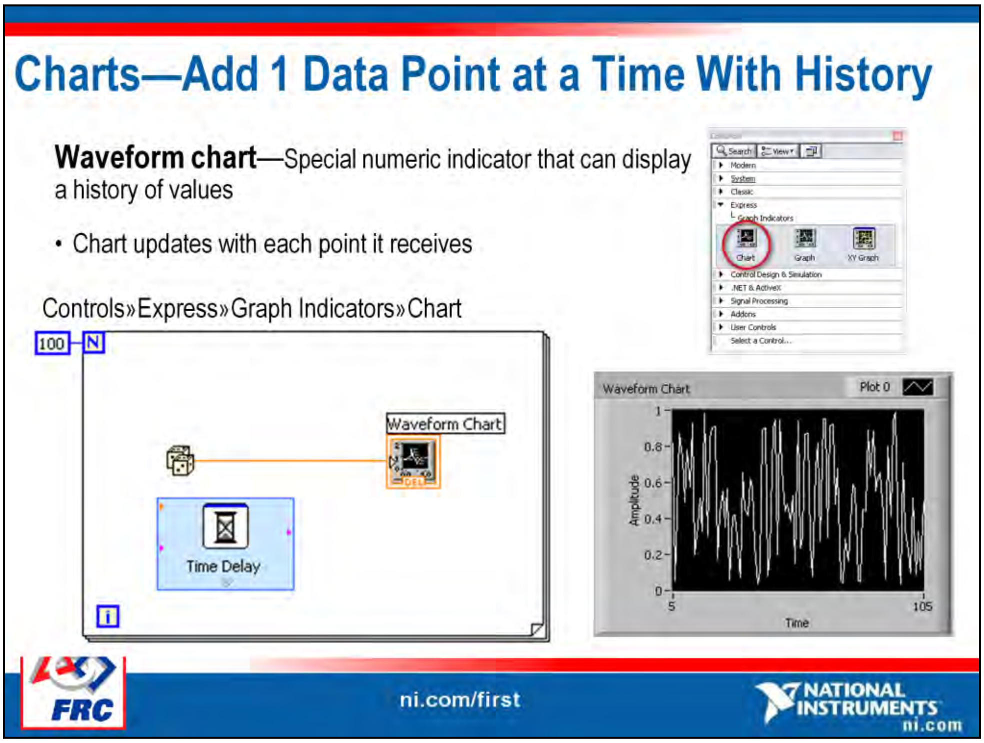
Timed Loop

Time Delay
The Time Delay Express VI delays execution by a specified number of seconds. Following the rules of dataflow programming, the While Loop will not iterate until all tasks inside it are complete, thus delaying each iteration of the loop.

Timed Loops
These loops execute each iteration of the loop at the period you specify. Use the Timed Loop when you want to develop VIs with multirate timing capabilities, precise timing, feedback-on-loop execution, timing characteristics that change dynamically, or several levels of execution priority.

Double-click the Input Node or right-click the Input Node and select **Configure Timed Loop** from the shortcut menu to display the Loop Configuration dialog box, where you can configure the Timed Loop. The values you enter in the **Loop Configuration** dialog box appear as options in the Input Node.

Wait Until Next ms Multiple
Waits until the value of the millisecond timer becomes a multiple of the specified millisecond multiple. Use this function to synchronize activities. You can call this function in a loop to control the loop execution rate. However, it is possible that the first loop period might be short. This function makes asynchronous system calls, but the nodes themselves function synchronously. Therefore, it does not complete execution until the specified time has elapsed.
Functions»Programming»Timing»Wait Until Next ms Multiple



Charts—Add 1 Data Point at a Time With History

Waveform chart—Special numeric indicator that can display a history of values

- Chart updates with each point it receives

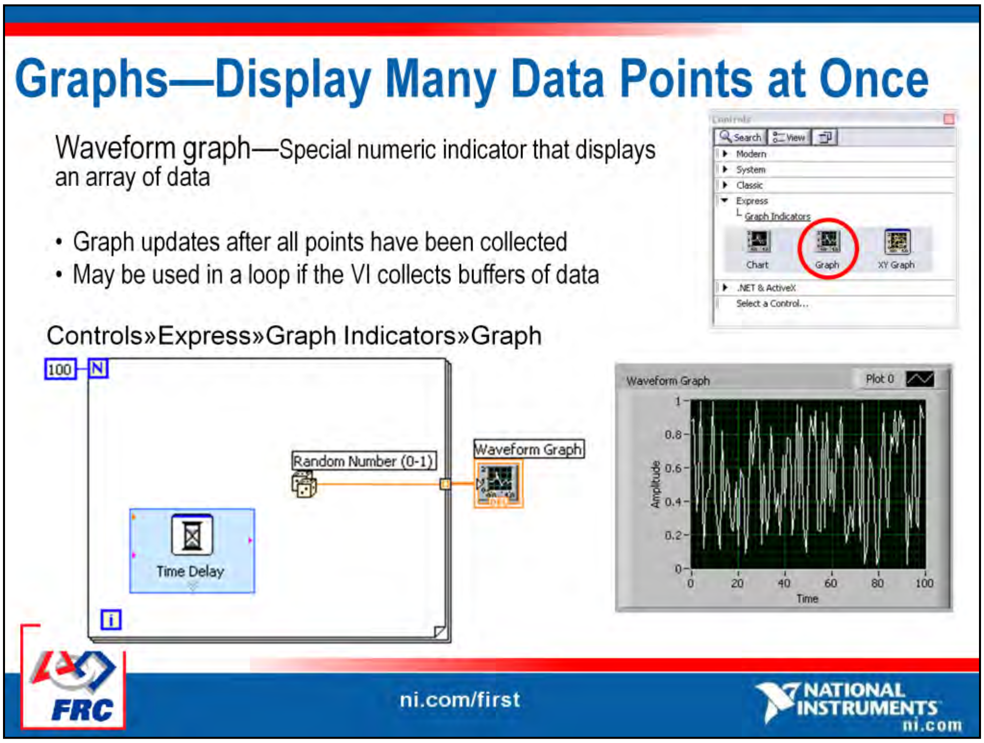
Controls»Express»Graph Indicators»Chart

ni.com/first

NATIONAL INSTRUMENTS ni.com

The waveform chart is a special numeric indicator that displays one or more plots. It is located on the **Controls»Modern»Graph** palette. Waveform charts can display single or multiple plots. The following front panel shows an example of a multiplot waveform chart.

You can change the minimum and maximum values of either the x-axis or y-axis by double-clicking on the value with the labeling tool and typing the new value. Similarly, you can change the label of the axis. You can also right-click the plot legend and change the style, shape, and color of the trace that is displayed on the chart.



Graphs—Display Many Data Points at Once

Waveform graph—Special numeric indicator that displays an array of data

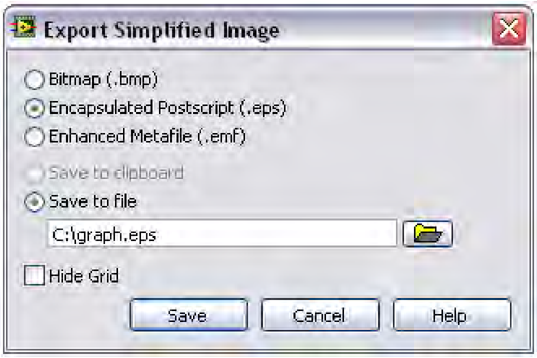
- Graph updates after all points have been collected
- May be used in a loop if the VI collects buffers of data

Controls»Express»Graph Indicators»Graph

Graphs are very powerful indicators in LabVIEW. You can use these highly customizable tools to concisely display a great deal of information.

With the properties page of the graph, you can display settings for plot types, scale and cursor options, and many other features of the graph. To open the properties page, right-click the graph on the front panel and choose **Properties.**

You can also create technical-paper-quality graphics with the "export simplified image" function. Right-click the graph and select **Data Operations»Export Simplified Image…**



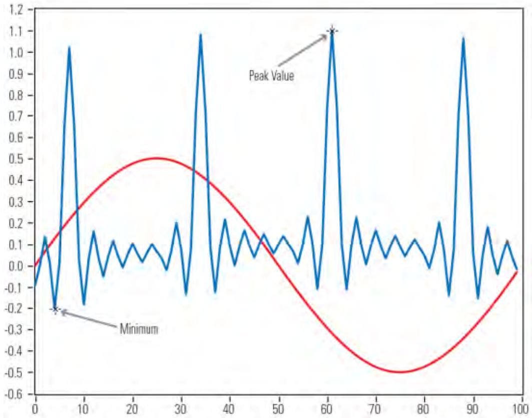
Export Simplified Image

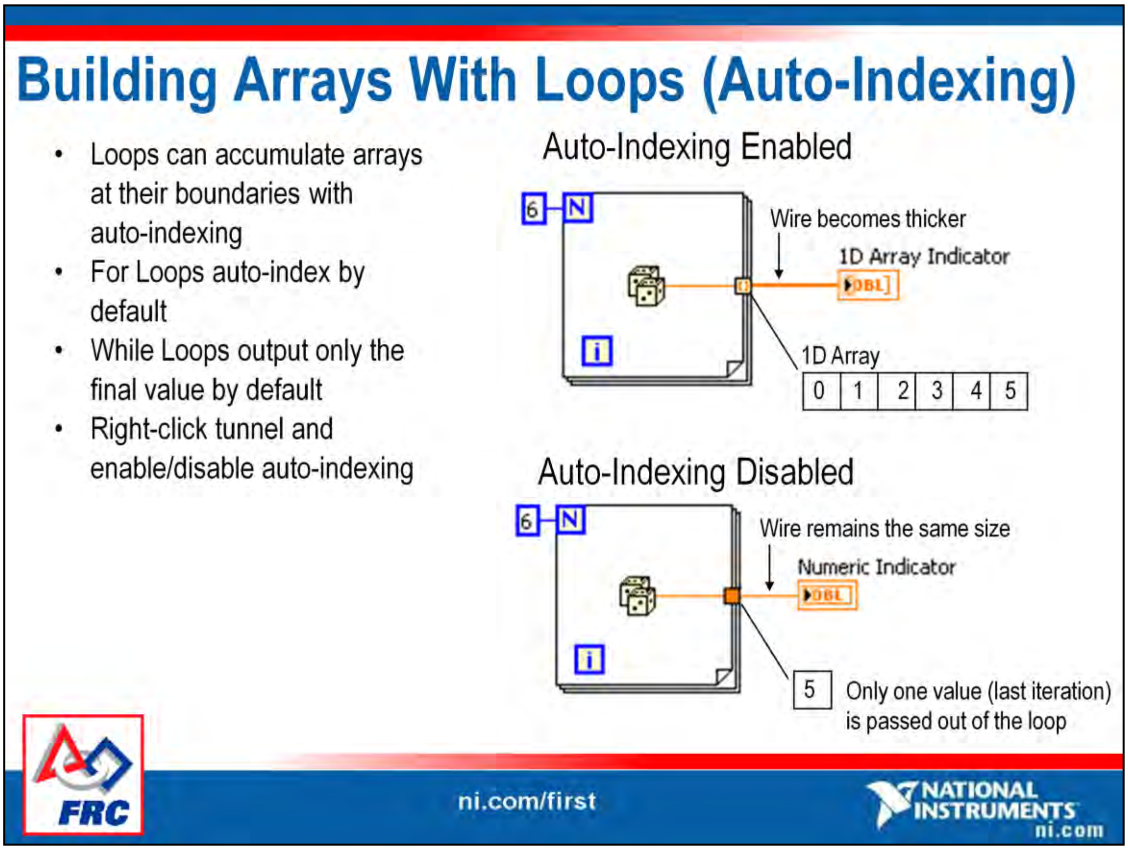
○ Bitmap (.bmp)
● Encapsulated Postscript (.eps)
○ Enhanced Metafile (.emf)

○ Save to clipboard
● Save to file

C:\graph.eps

□ Hide Grid





Building Arrays With Loops (Auto-Indexing)

- Loops can accumulate arrays at their boundaries with auto-indexing
- For Loops auto-index by default
- While Loops output only the final value by default
- Right-click tunnel and enable/disable auto-indexing

Auto-Indexing Enabled

Wire becomes thicker

1D Array Indicator

1D Array

Auto-Indexing Disabled

Wire remains the same size

Numeric Indicator

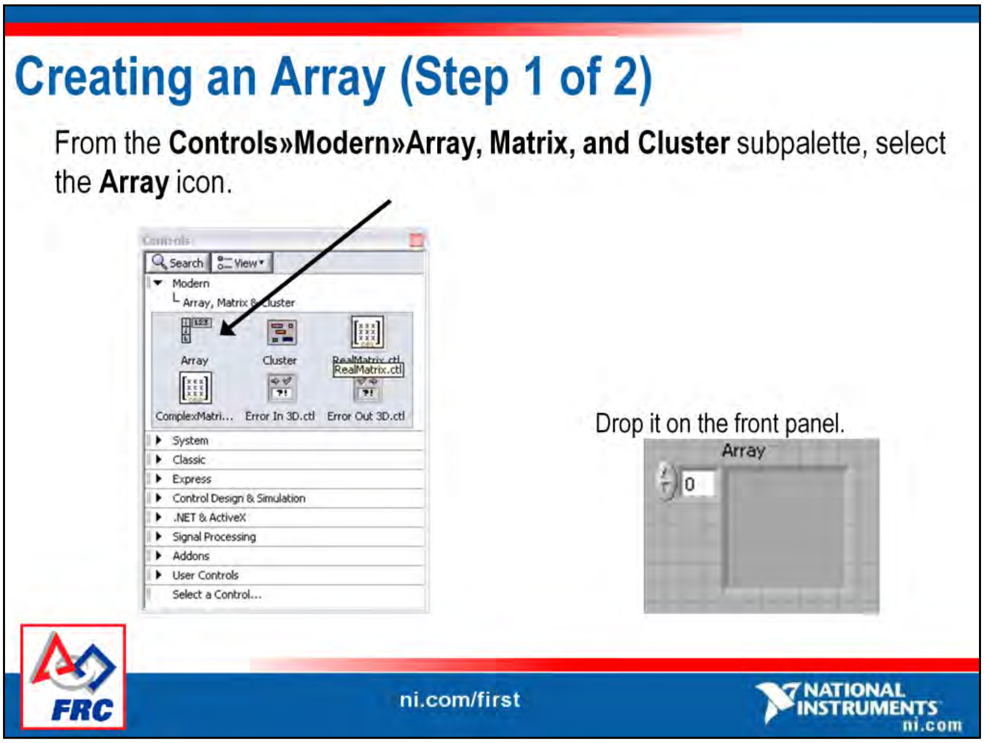
Only one value (last iteration) is passed out of the loop

For Loops and While Loops can index and accumulate arrays at their boundaries. This is known as auto-indexing.

- The indexing point on the boundary is called a tunnel
- The For Loop is auto-indexing-enabled by default
- The While Loop is auto-indexing-disabled by default

Examples

- Enable auto-indexing to collect values within the loop and build the array. All values are placed in the array upon exiting the loop.
- Disable auto-indexing if you are interested only in the final value.



Creating an Array (Step 1 of 2)

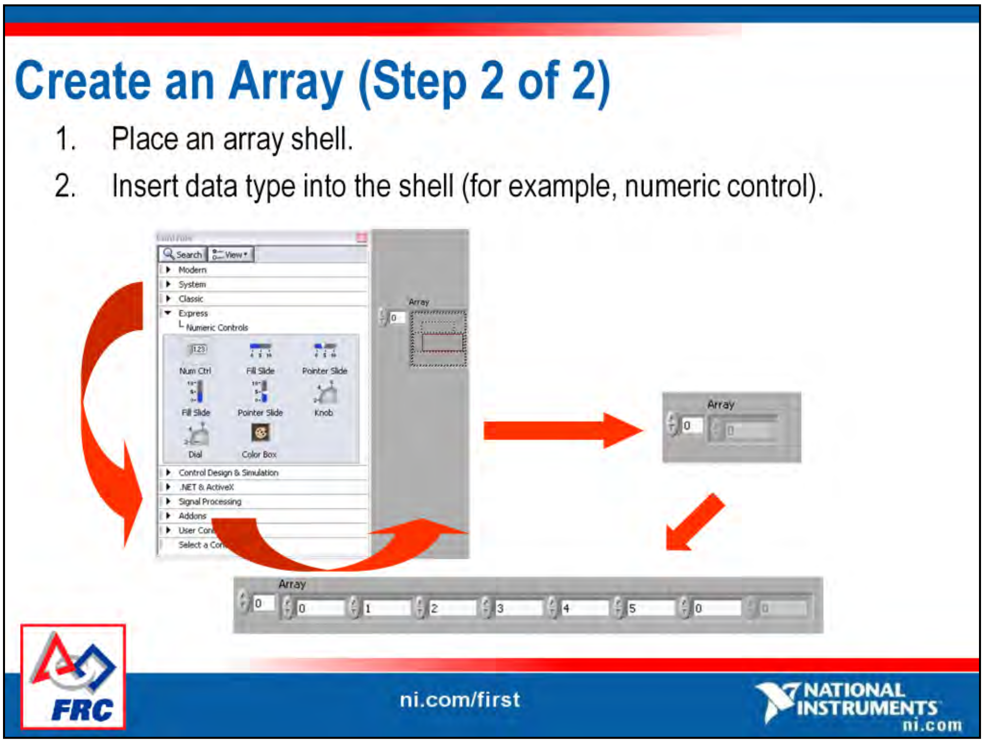
From the **Controls»Modern»Array, Matrix, and Cluster** subpalette, select the **Array** icon.

Drop it on the front panel.

ni.com/first

To create an array control or indicator as shown, select an array on the **Controls»Modern»Array, Matrix, and Cluster** palette, place it on the front panel, and drag a control or indicator into the array shell. If you attempt to drag an invalid control or indicator such as an XY graph into the array shell, you are unable to drop the control or indicator in the array shell.

You must insert an object in the array shell before you use the array on the block diagram. Otherwise, the array terminal appears black with an empty bracket.



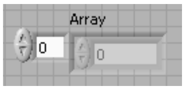
Create an Array (Step 2 of 2)

1. Place an array shell.
2. Insert data type into the shell (for example, numeric control).

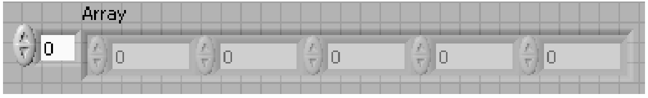
ni.com/first

To add dimensions to an array one at a time, right-click the index display and select **Add Dimension** from the shortcut menu. You also can use the Positioning tool to resize the index display until you have as many dimensions as you want.

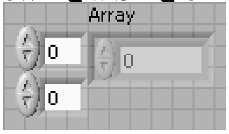
1D Array Viewing a Single Element:



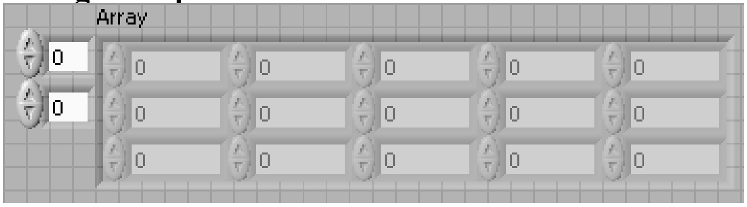
1D Array Viewing Multiple Elements:

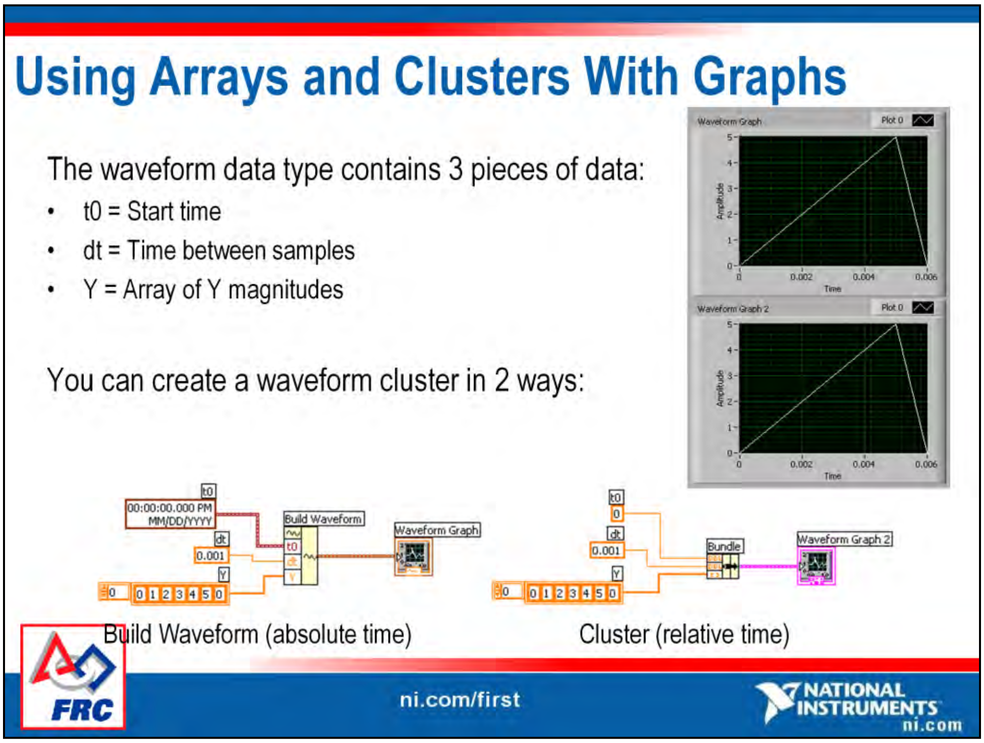


2D Array Viewing a Single Element:



2D Array Viewing Multiple Elements:





Using Arrays and Clusters With Graphs

The waveform data type contains 3 pieces of data:

- t0 = Start time
- dt = Time between samples
- Y = Array of Y magnitudes

You can create a waveform cluster in 2 ways:

Build Waveform (absolute time)

Cluster (relative time)

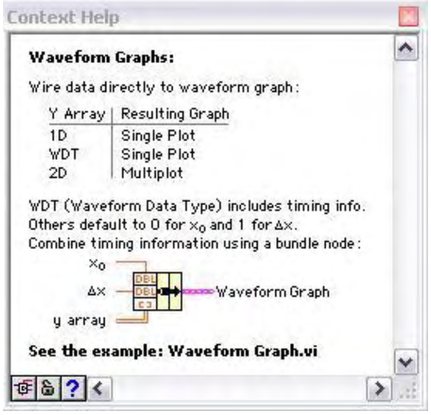
The waveform data type carries the data, start time, and Δ*t* of a waveform. You can create waveforms using the Build Waveform function. Many of the VIs and functions you use to acquire or analyze waveforms accept and return the waveform data type by default. When you wire a waveform data type to a waveform graph or chart, the graph or chart automatically plots a waveform based on the data, start time, and Δ*x* of the waveform. When you wire an array of waveform data types to a waveform graph or chart, the graph or chart automatically plots all of the waveforms.

Build Waveform

Builds a waveform or modifies an existing waveform with the start time represented as an absolute timestamp. Timestamps are accurate to real-world time and date and are very useful for real-world data recording.

Bundle

Builds a waveform or modifies an existing waveform with a relative timestamp. The input to t_0 is a DBL. By building waveforms using the bundle, you can plot data on the negative x-axis (time).



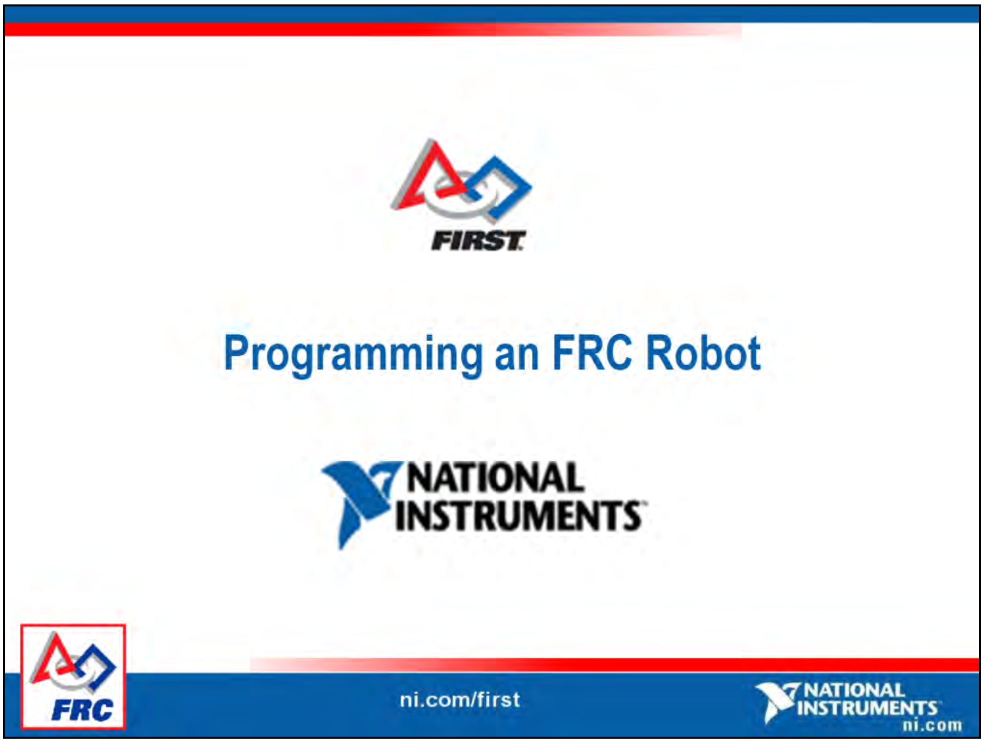
Waveform Graphs:

Wire data directly to waveform graph:

Y Array	Resulting Graph
1D	Single Plot
WDT	Single Plot
2D	Multiplot

WDT (Waveform Data Type) includes timing info. Others default to 0 for x_0 and 1 for Δx. Combine timing information using a bundle node:

See the example: Waveform Graph.vi



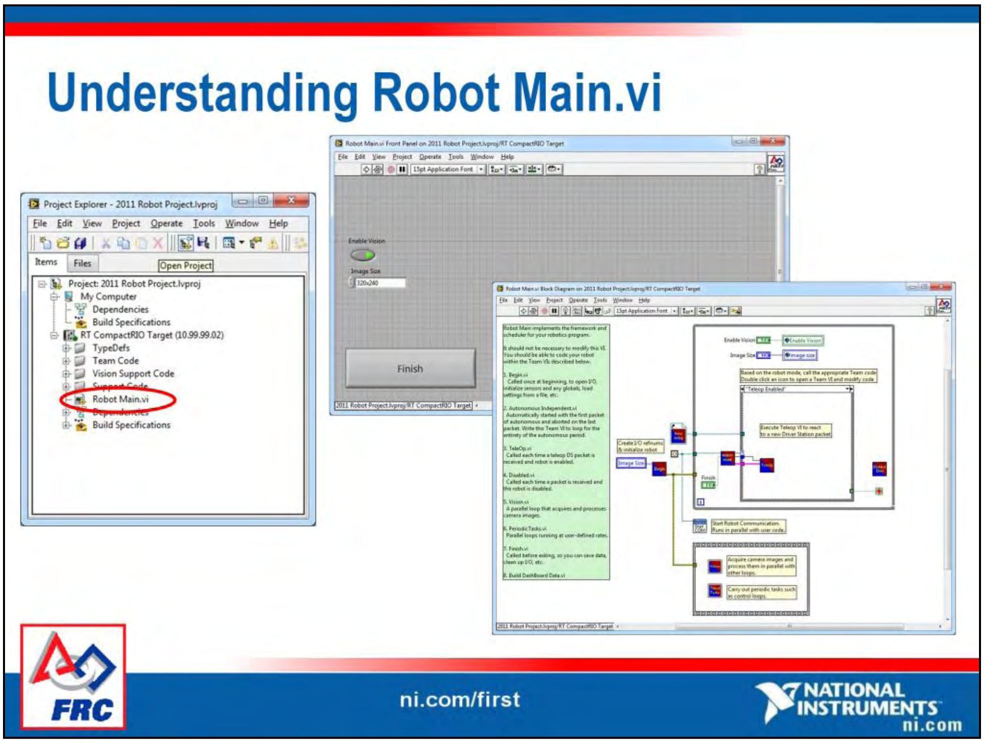
Programming an FRC Robot

ni.com/first

Section Objectives:

A. Understand the FRC code template including structure components, subVIs.

B. Program a robot by completing several exercises.



Understanding Robot Main.vi

Project Explorer - 2011 Robot Project.lvproj
File Edit View Project Operate Tools Window Help
Items Files
Open Project
Project: 2011 Robot Project.lvproj
My Computer
Dependencies
Build Specifications
RT CompactRIO Target (10.99.99.02)
TypeDefs
Team Code
Vision Support Code
Support Code
Robot Main.vi
Dependencies
Build Specifications
Robot Main.vi Front Panel on 2011 Robot Project.lvproj/RT CompactRIO Target
15pt Application Font
Enable Vision
Image Size
320x240
Finish
2011 Robot Project.lvproj/RT CompactRIO Target
Robot Main.vi Block Diagram on 2011 Robot Project.lvproj/RT CompactRIO Target
Enable Vision
Image Size
"Teleop Enabled"
Finish
Begin
TeleOp

Robot Main implements the framework and scheduler for your robotics program.

It should not be necessary to modify this VI. You should be able to code your robot within the Team VIs described below.

1. Begin.vi
Called once at beginning, to open I/O, initialize sensors and any globals, load settings from a file, etc.

2. Autonomous Independent.vi
Automatically started with the first packet of autonomous and aborted on the last packet. Write this Team VI to loop for the entirety of the autonomous period.

3. TeleOp.vi
Called each time a teleop DS packet is received and robot is enabled.

4. Disabled.vi
Called each time a packet is received and the robot is disabled.

5. Vision.vi
A parallel loop that acquires and processes camera images.

6. PeriodicTasks.vi
Parallel loops running at user-defined rates.

7. Finish.vi
Called before exiting, so you can save data, clean up I/O, etc.

8. Build DashBoard Data.vi

Based on the robot mode, call the appropriate Team code
Double click an icon to open a Team VI and modify code

Execute Teleop VI to react to a new Driver Station packet

Create I/O refnums & initialize robot

Start Robot Communication. Runs in parallel with user code.

Acquire camera images and process them in parallel with other loops.

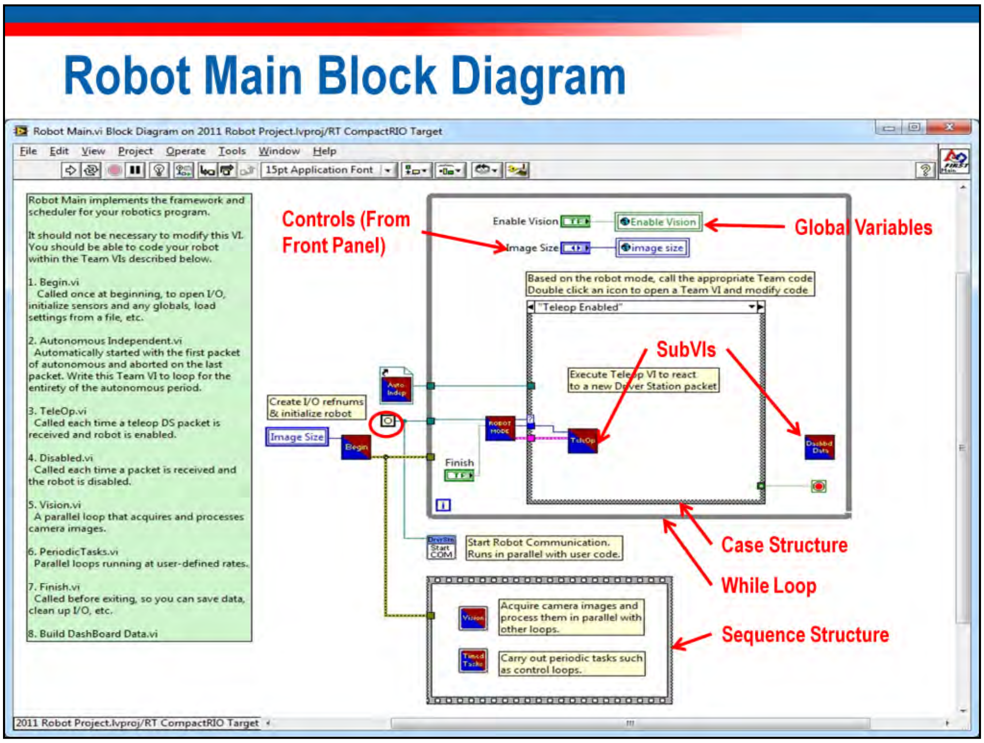
Carry out periodic tasks such as control loops.

FRC

ni.com/first

NATIONAL INSTRUMENTS
ni.com

As you will see in the next exercise, when you configure a new project in LabVIEW FRC, a default code template will be available. By using this core template, you should be able to drive the robot in Arcade mode without making any code changes. In the coming slides, we will investigate the block diagram (code) of the Robot Main.vi to understand the key components and their functions.



Robot Main Block Diagram

Robot Main.vi Block Diagram on 2011 Robot Project.lvproj/RT CompactRIO Target

Robot Main implements the framework and scheduler for your robotics program.

It should not be necessary to modify this VI. You should be able to code your robot within the Team VIs described below.

1. Begin.vi
 Called once at beginning, to open I/O, initialize sensors and any globals, load settings from a file, etc.

2. Autonomous Independent.vi
 Automatically started with the first packet of autonomous and aborted on the last packet. Write this Team VI to loop for the entirety of the autonomous period.

3. TeleOp.vi
 Called each time a teleop DS packet is received and robot is enabled.

4. Disabled.vi
 Called each time a packet is received and the robot is disabled.

5. Vision.vi
 A parallel loop that acquires and processes camera images.

6. PeriodicTasks.vi
 Parallel loops running at user-defined rates.

7. Finish.vi
 Called before exiting, so you can save data, clean up I/O, etc.

8. Build DashBoard Data.vi

Based on the robot mode, call the appropriate Team code
Double click an icon to open a Team VI and modify code

Execute Teleop VI to react
to a new Driver Station packet

Create I/O refnums
& initialize robot

Start Robot Communication.
Runs in parallel with user code.

Acquire camera images and
process them in parallel with
other loops.

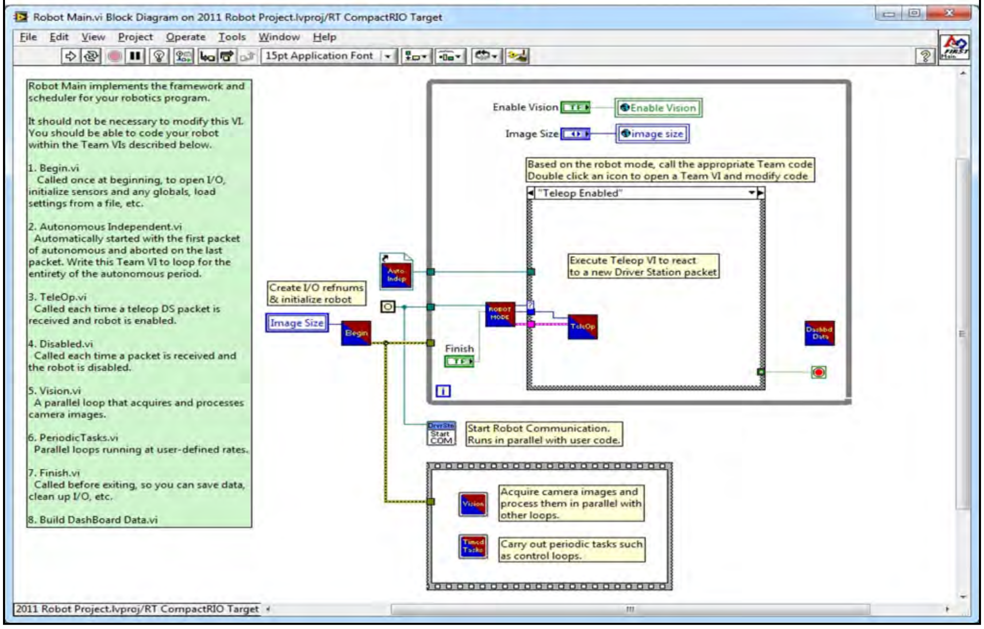
Carry out periodic tasks such
as control loops.

2011 Robot Project.lvproj/RT CompactRIO Target

Robot Main implements the framework and scheduler for your robotics program.
It should not be necessary to modify this VI. You should be able to code your robot within the Team VIs described below.

1. Begin.vi
Called once at beginning, to open I/O, initialize sensors and any globals, load settings from a file, and so on.
2. Autonomous Independent.vi
Automatically started with the first packet of autonomous and aborted on the last packet. Write this Team VI to loop for the entirety of the autonomous period.
3. TeleOp.vi
Called each time a teleop DS packet is received and robot is enabled.
4. Disabled.vi
Called each time a packet is received and the robot is disabled.
5. Vision.vi
A parallel loop that acquires and processes camera images.
6. PeriodicTasks.vi
Parallel loops running at user-defined rates.
7. Finish.vi
Called before exiting, so you can save data, clean up I/O, etc.
8. Build DashBoard Data.vi
Called for each driver station packet including timeouts. It is used to send select I/O values back to the Dashboard.

Dataflow Programming



Robot Main implements the framework and scheduler for your robotics program.

It should not be necessary to modify this VI. You should be able to code your robot within the Team VIs described below.

1. Begin.vi
Called once at beginning, to open I/O, initialize sensors and any globals, load settings from a file, etc.

2. Autonomous Independent.vi
Automatically started with the first packet of autonomous and aborted on the last packet. Write this Team VI to loop for the entirety of the autonomous period.

3. TeleOp.vi
Called each time a teleop DS packet is received and robot is enabled.

4. Disabled.vi
Called each time a packet is received and the robot is disabled.

5. Vision.vi
A parallel loop that acquires and processes camera images.

6. PeriodicTasks.vi
Parallel loops running at user-defined rates.

7. Finish.vi
Called before exiting, so you can save data, clean up I/O, etc.

8. Build DashBoard Data.vi

Based on the robot mode, call the appropriate Team code
Double click an icon to open a Team VI and modify code

Execute Teleop VI to react
to a new Driver Station packet

Create I/O refnums
& initialize robot

Start Robot Communication.
Runs in parallel with user code.

Acquire camera images and process them in parallel with other loops.

Carry out periodic tasks such as control loops.

Referencing the LabVIEW block diagram above, the code will start by executing the Begin command, the Reference to the Autonomous Independent.vi, and the Generate Occurrence.vi in parallel.

The StartCOM.vi will execute sequentially after the Generate Occurrence.vi has executed since it is connected with a data wire to the Generate Occurrence.vi.

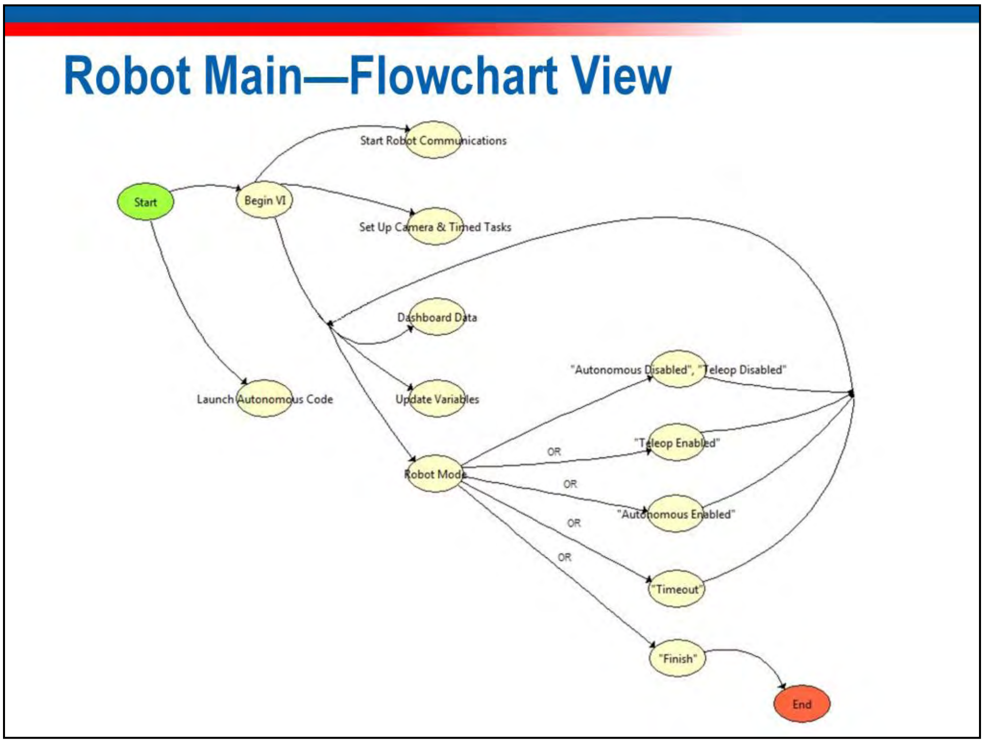
The Sequence Structure with the Vision.vi will execute sequentially after the Begin command has been executed since it is connected to the Begin.vi with an error wire.

The large While Loop will execute only after the Begin.vi, Generate Occurrence.vi, and The Reference to the Autonomous Independent.vi have all executed. Data from these VIs must all flow to the edge of the While Loop before the While Loop will start. This is because the While Loop is waiting for data from all of these VIs as indicated by the wires connected to the left side of the While Loop. (The side of the loop the wire are connected is not important, but mentioned for clarity in referencing the image above.)

The Sequence Structure and the While Loop will be running in parallel.

Within the Sequence Structure, the Vision.vi and Timed Tasks.vi run in parallel. This is because there are no data wires connecting them to control the execution order.

Within the While Loop, the Robot Mode.vi will run in parallel to the Dashboard Data.vi. The TeleOp.vi will run sequentially after the Robot Mode.vi since they are connected with data wires.



Robot Main—Flowchart View

Start

Begin VI

Start Robot Communications

Set Up Camera & Timed Tasks

Dashboard Data

Update Variables

Launch Autonomous Code

Robot Mode

OR

OR

OR

OR

"Autonomous Disabled", "Teleop Disabled"

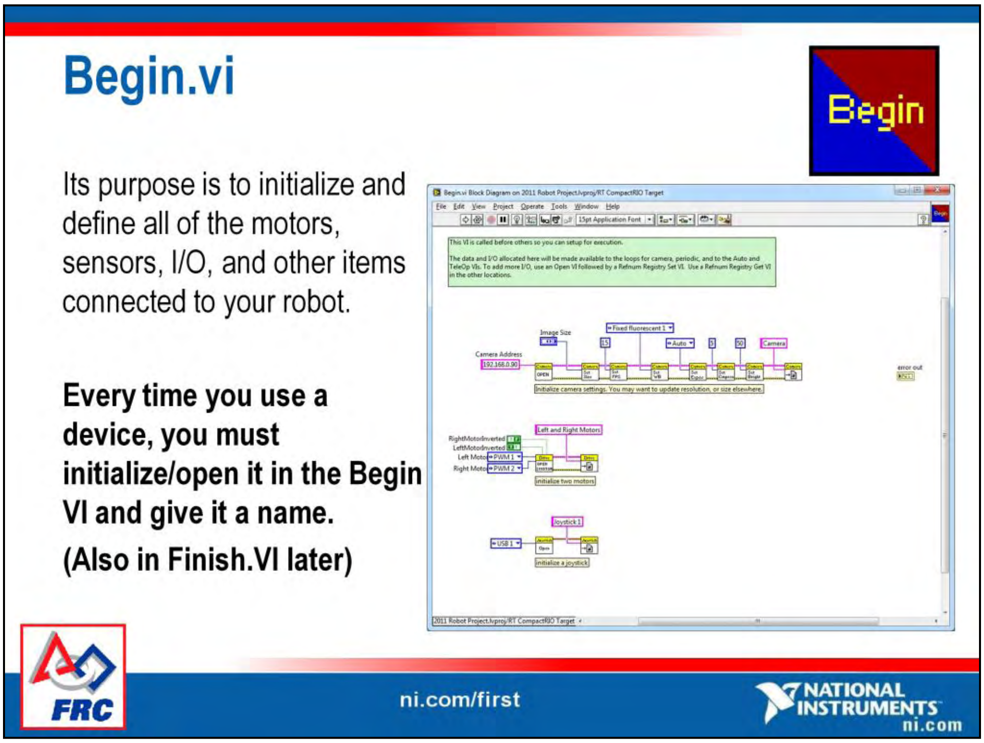
"Teleop Enabled"

"Autonomous Enabled"

"Timeout"

"Finish"

End



Begin.vi

Its purpose is to initialize and define all of the motors, sensors, I/O, and other items connected to your robot.

Every time you use a device, you must initialize/open it in the Begin VI and give it a name.

(Also in Finish.VI later)

Begin.vi Block Diagram on 2011 Robot Project.lvproj/RT CompactRIO Target

This VI is called before others so you can setup for execution.

The data and I/O allocated here will be made available to the loops for camera, periodic, and to the Auto and TeleOp VIs. To add more I/O, use an Open VI followed by a Refnum Registry Set VI. Use a Refnum Registry Get VI in the other locations.

Initialize camera settings. You may want to update resolution, or size elsewhere.

initialize two motors

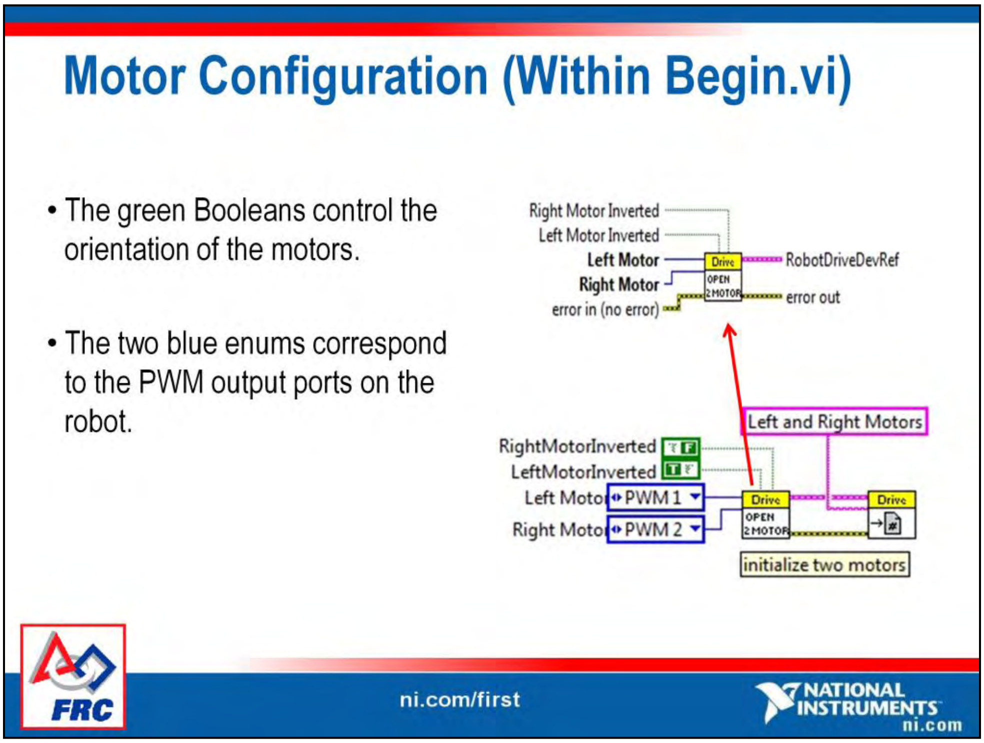
initialize a joystick

This subVI is called once at the start of the Robot Main.vi.

Its purpose is to initialize and define all of the motors, sensors, I/O, and other items connected to your robot.

The default FRC robot code initializes the camera settings, two motors, and joystick.

Turn on the **Context Help** by pressing <**Ctrl-H**>. Then hover over each function on the block diagram to see its name and description.



Motor Configuration (Within Begin.vi)

- The green Booleans control the orientation of the motors.
- The two blue enums correspond to the PWM output ports on the robot.

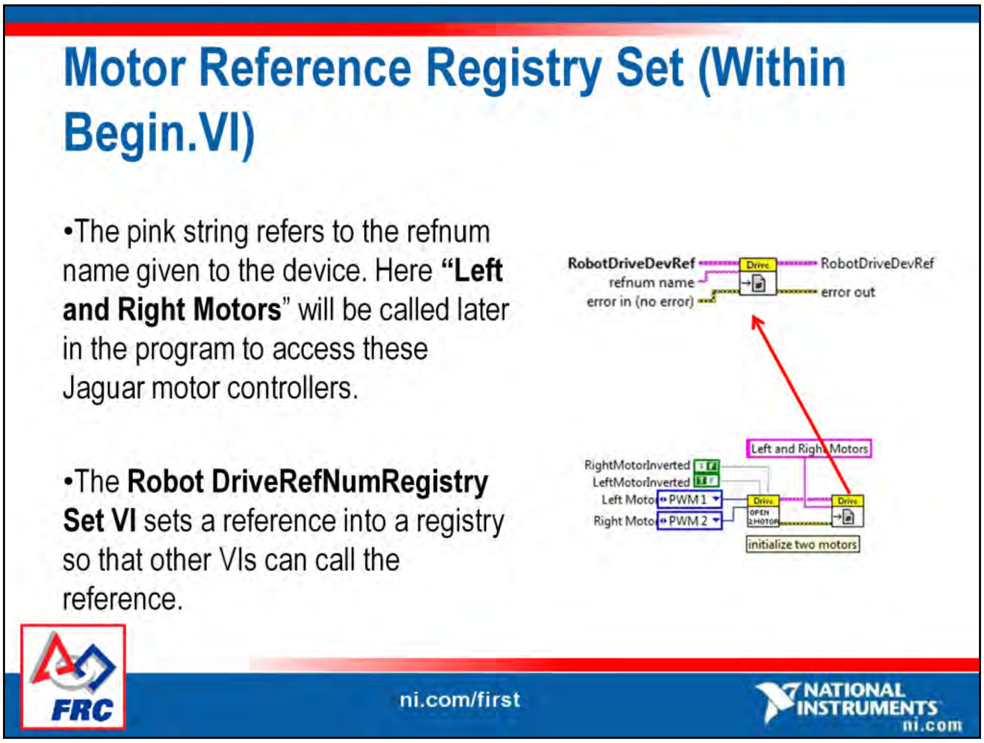
ni.com/first

There are two green Boolean constants wired into the Open 2 Motor VI. They control the orientation of the motors.

There are two blue parameters wired into Open 2 Motor VI, PWM 1 and PWM 2. They correspond to the PWM output ports on the robot.

The Open 2 Motor VI outputs a reference cluster and an error cluster to the Motor Reference Registry Set VI.

The Open 2 Motors VI opens a reference to the robot drive for a two-motor robot using Jaguar motor controllers.



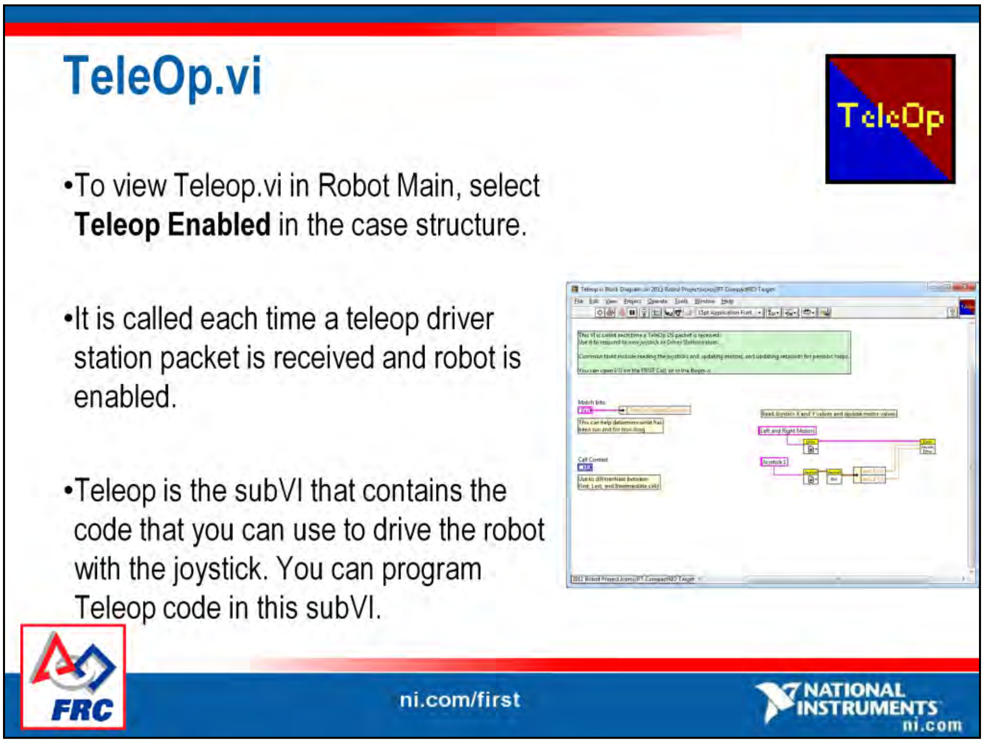
Motor Reference Registry Set (Within Begin.VI)

- The pink string refers to the refnum name given to the device. Here **"Left and Right Motors"** will be called later in the program to access these Jaguar motor controllers.
- The **Robot DriveRefNumRegistry Set VI** sets a reference into a registry so that other VIs can call the reference.

Motor Reference Registry Set VI name is Robot DriveRefNumRegistry Set VI.

The pink string refers to the refnum name given to the device. Here "Left and Right Motors" will be called later in the program when you need to access these Jaguar motor controllers.

The **Drive Reference Registry Set VI** sets a reference into a registry so that other VIs can call the reference.



TeleOp.vi

- To view Teleop.vi in Robot Main, select **Teleop Enabled** in the case structure.
- It is called each time a teleop driver station packet is received and robot is enabled.
- Teleop is the subVI that contains the code that you can use to drive the robot with the joystick. You can program Teleop code in this subVI.

Teleop.vi Block Diagram on 2011 Robot Project.lvproj/RT CompactRIO Target

This VI is called each time a TeleOp DS packet is received.
Use it to respond to new joystick or Driver Station values.

Common tasks include reading the joysticks and updating motors, and updating setpoints for periodic loops.

You can open I/O on the FIRST Call, or in the Begin.vi.

This can help determine what has been run and for how long.

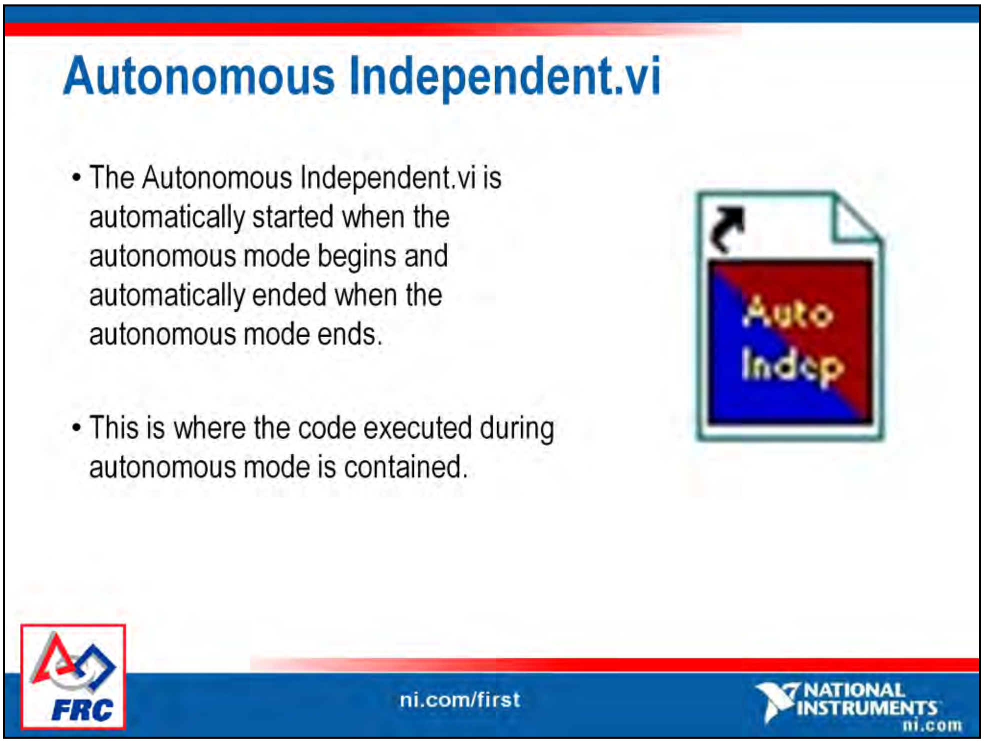
Use to differentiate between First, Last, and Intermediate calls.

Read Joystick X and Y values and update motor values.

To see the block diagram of the Teleop SubVI, select Teleop Enable in the case structure and then double-click it in the Robot Main. The front panel of the Teleop SubVI will open.

Switch to its block diagram.

This is the code that reads the joystick and makes the robot move in Arcade Drive.



Autonomous Independent.vi

- The Autonomous Independent.vi is automatically started when the autonomous mode begins and automatically ended when the autonomous mode ends.
- This is where the code executed during autonomous mode is contained.

Auto
Indep
FRC
ni.com/first
NATIONAL INSTRUMENTS
ni.com

Autonomous mode is an Independent SubVI that gets launched whenever the robot driver station instructs the robot to go into autonomous mode.

Unlike the Teleop VI, the autonomous independent VI actually runs once but lasts for several seconds. It does not run over and over again. Therefore, the Autonomous Independent SubVI is not included in the main loop.

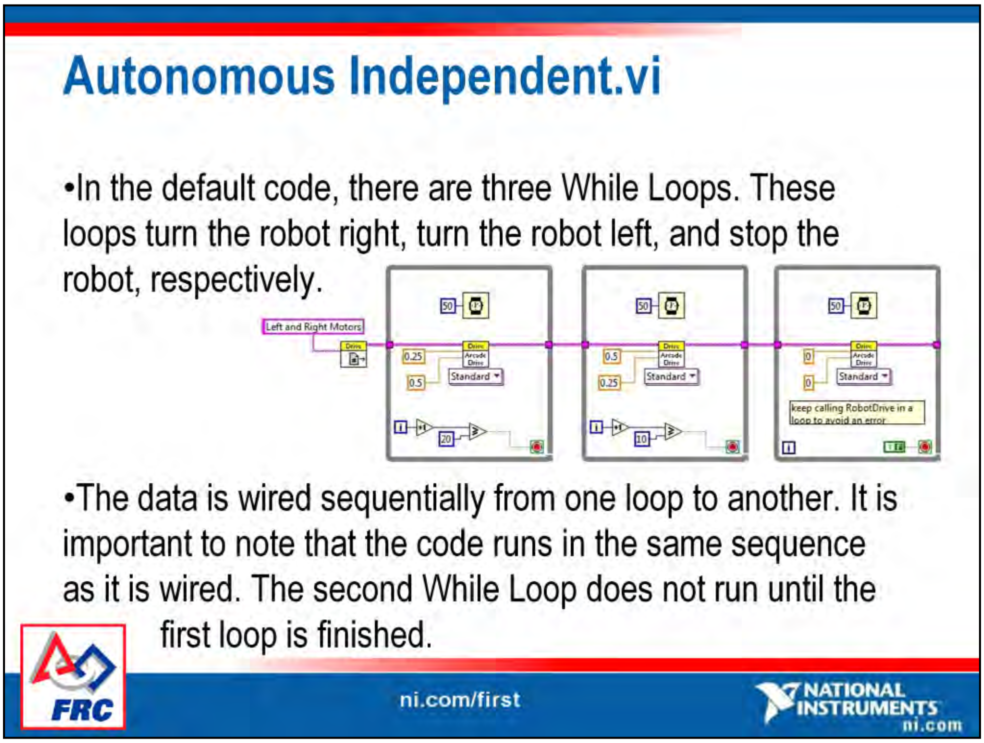
To access the Autonomous Independent SubVI , double-click on the SubVI reference icon in the Robot Main.vi.

Switch to the block diagram of the Autonomous Independent SubVI by pressing <Ctrl-E>.

On the left of the block diagram, there are Driver Station SubVIs that provide data on driver position, alliance color, digital switch settings, and analog voltages.

The main code is disabled to prevent unexpected execution in Autonomous mode.

To enable this code, right-click on the edge of the frame and choose **Enable This Subdiagram.**



Autonomous Independent.vi

•In the default code, there are three While Loops. These loops turn the robot right, turn the robot left, and stop the robot, respectively.

•The data is wired sequentially from one loop to another. It is important to note that the code runs in the same sequence as it is wired. The second While Loop does not run until the first loop is finished.

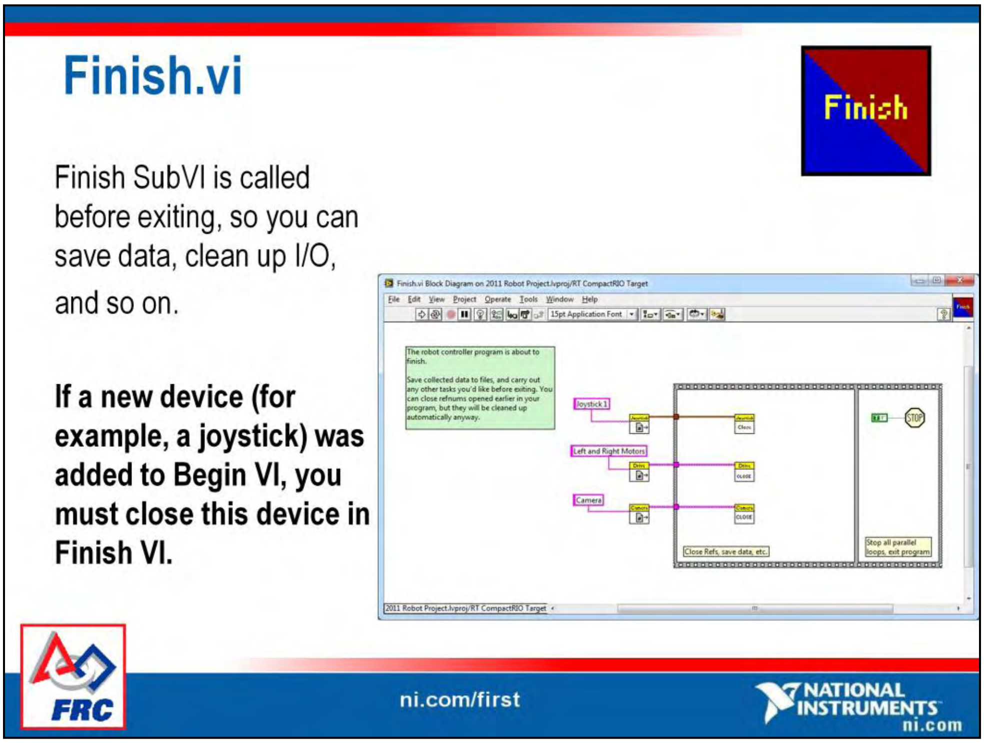
In the first loop, the code will turn the robot to the right for 1 second. The loop iterates every 50 ms because of the **wait (ms)** function. (50 ms * 20 loops = 1 second)

The loop will **run 20 times** because there is a comparison function attached to the iteration terminal.

1 is added to the value of the iteration terminal because it begins counting at 0. When the While Loop has iterated 20 times, the comparison function will output at true into the conditional terminal and will cause the program to exit the loop. The Arcade Drive VI in the first loop turns the robot to the right.

The second loop is similar to the first loop, but it will turn the robot to the left for 0.5 seconds. The loop iterates every 50 ms due to the milliseconds to wait function, and the loop will run 10 times, for 0.5 seconds in total. The Arcade Drive VI in the second loop turns the robot to the left.

The third loop stops the robot. This loop is an infinite loop because it has a false value (constant) wired to the conditional terminal, which causes the loop to run forever. The input values to the Arcade Drive VI are 0, which means the motors stand still.



Finish.vi

Finish SubVI is called before exiting, so you can save data, clean up I/O, and so on.

If a new device (for example, a joystick) was added to Begin VI, you must close this device in Finish VI.

Finish.vi Block Diagram on 2011 Robot Project.lvproj/RT CompactRIO Target

The robot controller program is about to finish.

Save collected data to files, and carry out any other tasks you'd like before exiting. You can close refnums opened earlier in your program, but they will be cleaned up automatically anyway.

Close Refs, save data, etc.

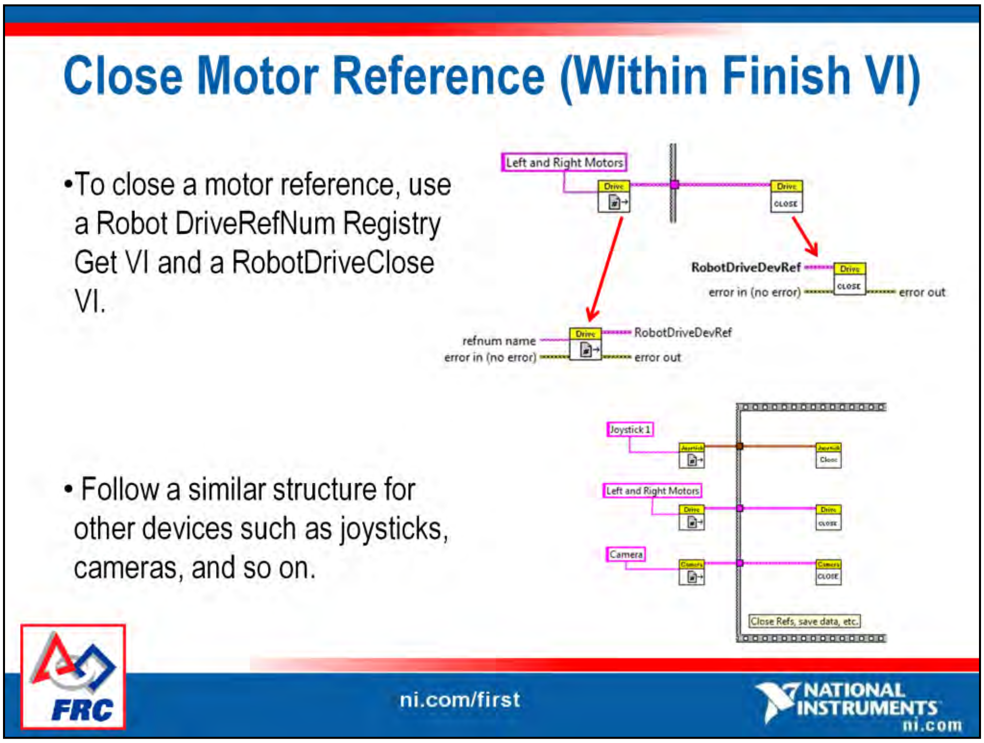
Stop all parallel loops, exit program

2011 Robot Project.lvproj/RT CompactRIO Target

To see the block diagram of Finish SubVI, select Finish in the case structure and then double-click it in Robot Main.

The default FRC robot code closes the reference of the camera settings, two motors, and joystick.

Whenever a device is added to Begin.vi, you should also close that device in Finish.vi.



Close Motor Reference (Within Finish VI)

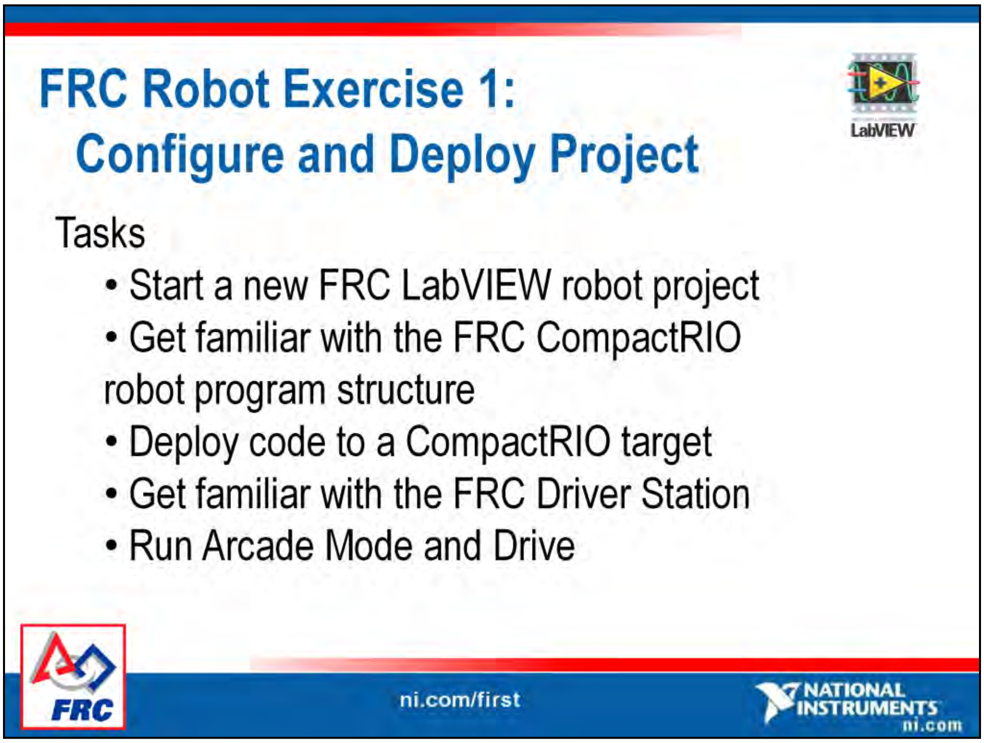
- To close a motor reference, use a Robot DriveRefNum Registry Get VI and a RobotDriveClose VI.

- Follow a similar structure for other devices such as joysticks, cameras, and so on.

To close a motor reference, use a Drive Reference Get VI (Robot DriveRefNum Registry Get.vi) and a Drive Reference Close VI (RobotDriveClose.vi).

The Drive Reference Get VI gets the drive reference previously created (in the Begin VI) by name and outputs a reference cluster to the Drive Reference Close VI.

The Drive Reference Close VI gets the reference cluster and closes the specific drive reference.



FRC Robot Exercise 1: Configure and Deploy Project

Tasks

- Start a new FRC LabVIEW robot project
- Get familiar with the FRC CompactRIO robot program structure
- Deploy code to a CompactRIO target
- Get familiar with the FRC Driver Station
- Run Arcade Mode and Drive

ni.com/first

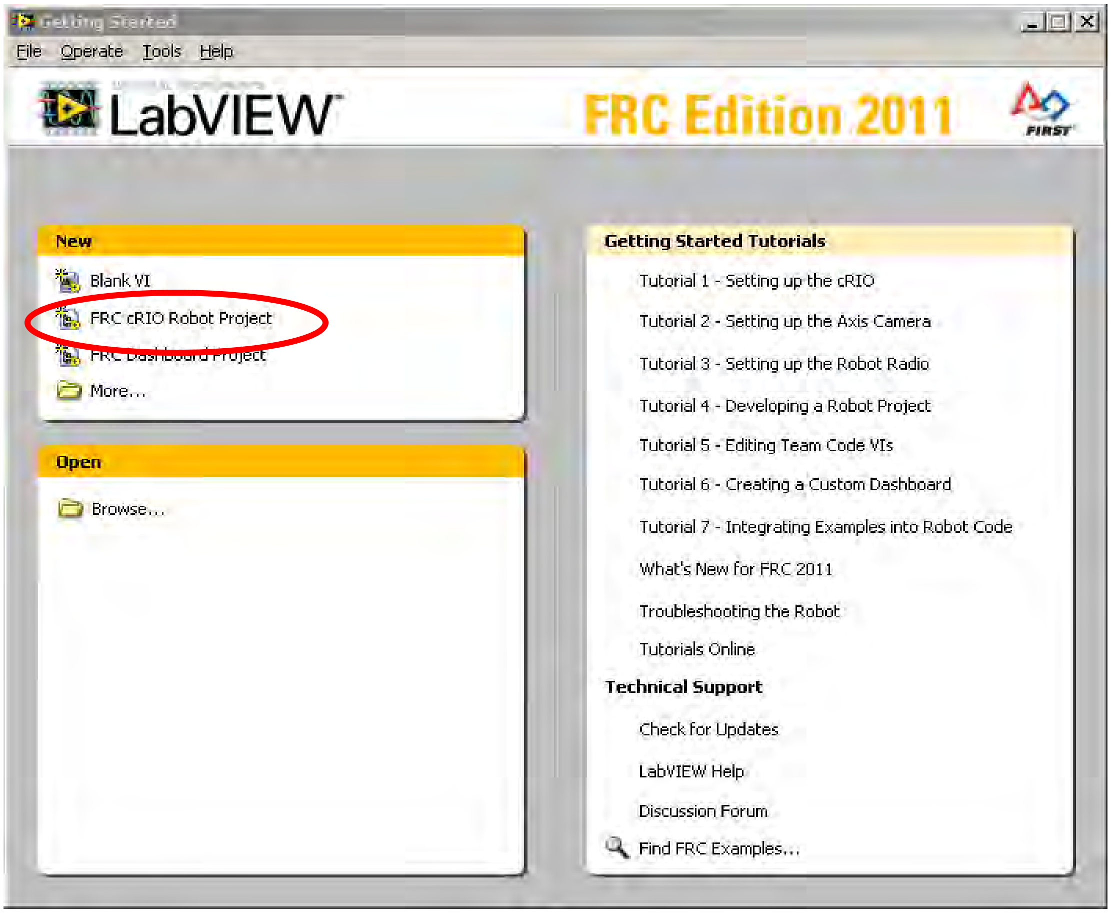
FRC Robot Exercise 1: Configure and Deploy Project

In this exercise, configure and deploy a LabVIEW Project and run the FRC Driver Station to drive the robot in arcade mode.

When you open the LabVIEW program, a Getting Started Window pops up.

Getting started tutorials for FRC are provided on the right side of the Getting Started Window. The tutorials include plenty of useful information.

1. To start a new FRC LabVIEW project, click **FRC cRIO Robot Project** on the left side of the Window.

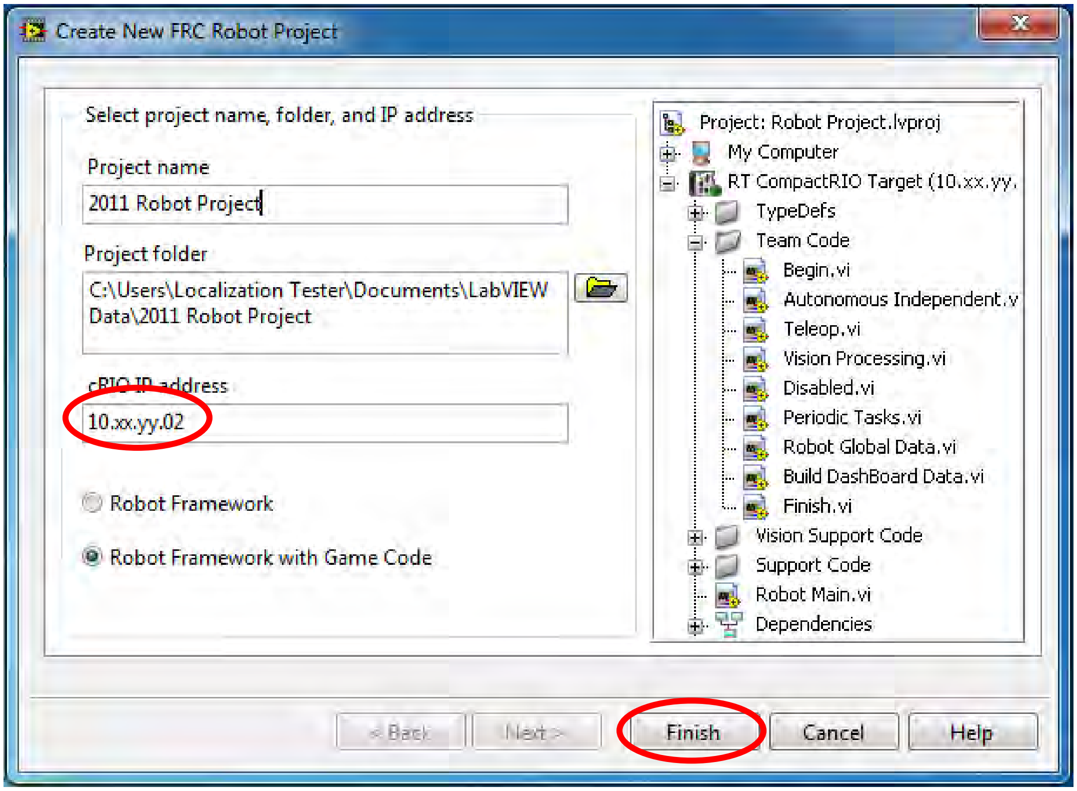

Getting Started
File Operate Tools Help
LabVIEW
FRC Edition 2011
FIRST
New
Blank VI
FRC cRIO Robot Project
FRC Dashboard Project
More...
Open
Browse...
Getting Started Tutorials
Tutorial 1 - Setting up the cRIO
Tutorial 2 - Setting up the Axis Camera
Tutorial 3 - Setting up the Robot Radio
Tutorial 4 - Developing a Robot Project
Tutorial 5 - Editing Team Code VIs
Tutorial 6 - Creating a Custom Dashboard
Tutorial 7 - Integrating Examples into Robot Code
What's New for FRC 2011
Troubleshooting the Robot
Tutorials Online
Technical Support
Check for Updates
LabVIEW Help
Discussion Forum
Find FRC Examples...

2. **Create New FRC cRIO Robot Project**

   Here, you can change
   - Project name
   - Project location
   - cRIO IP address   *Note: This setting tell LabVIEW the IP address of the cRIO control but does not set the IP address. To set the IP address use the cRIO Imaging Tool.

3. Replace xx.yy in the **cRIO IP address** with your four-digit team number.
    - Team #1234 → 10.**12.34**.02
    - Team #123 → 10.**01.23**.02

4. A preview of the project file is shown below.



Create New FRC Robot Project

Select project name, folder, and IP address

Project name: 2011 Robot Project

Project folder: C:\Users\Localization Tester\Documents\LabVIEW Data\2011 Robot Project

cRIO IP address: 10.xx.yy.02

○ Robot Framework

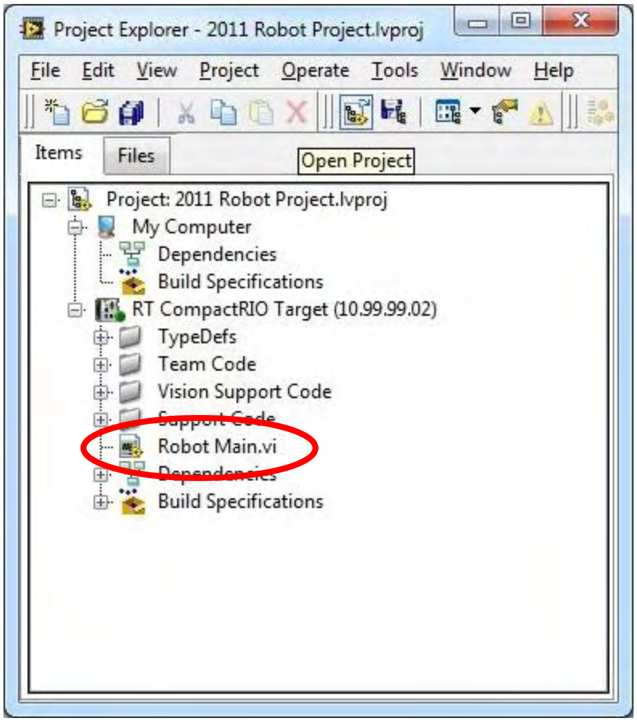
● Robot Framework with Game Code

```
Project: Robot Project.lvproj
  My Computer
  RT CompactRIO Target (10.xx.yy.
    TypeDefs
    Team Code
      Begin.vi
      Autonomous Independent.v
      Teleop.vi
      Vision Processing.vi
      Disabled.vi
      Periodic Tasks.vi
      Robot Global Data.vi
      Build DashBoard Data.vi
      Finish.vi
    Vision Support Code
    Support Code
    Robot Main.vi
    Dependencies
```

5. Click **Finish** when done.

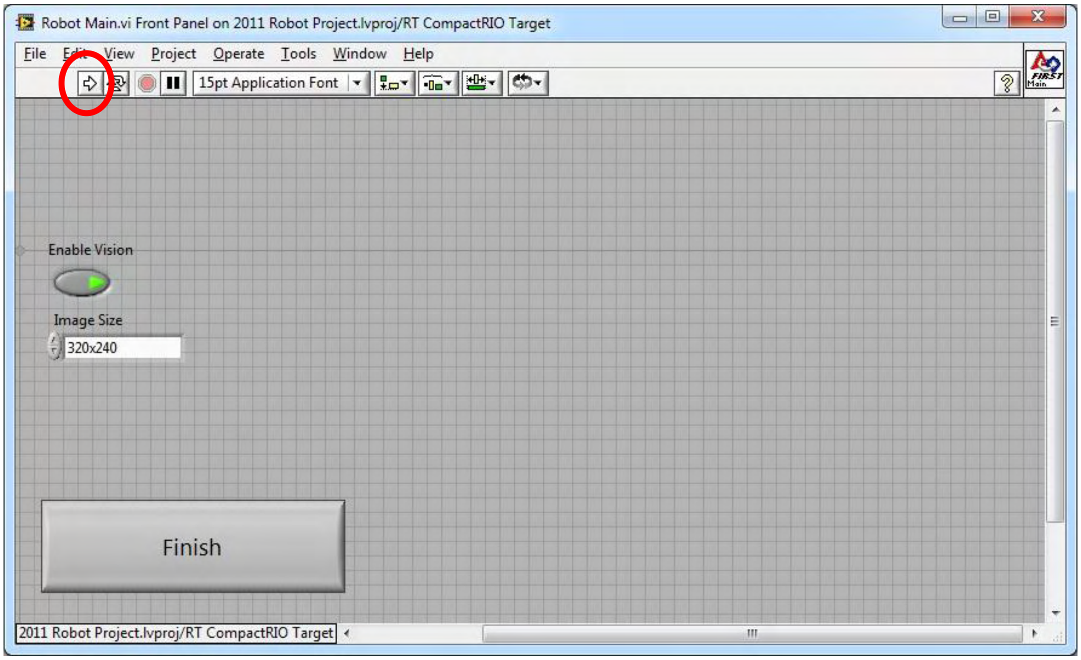
6. The Project Explorer opens after creating the new FRC robot project.
Note: You can use the Project Explorer Window to create and edit LabVIEW projects. All code included in the FRC robot project default template is listed under the **Items** tab.

7. To start modifying the robot code, expand RT CompactRIO Target and double-click **Robot Main.vi.**

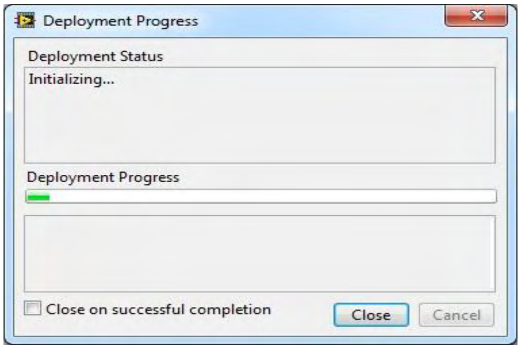

Project Explorer - 2011 Robot Project.lvproj
File Edit View Project Operate Tools Window Help
Items
Files
Open Project
Project: 2011 Robot Project.lvproj
My Computer
Dependencies
Build Specifications
RT CompactRIO Target (10.99.99.02)
TypeDefs
Team Code
Vision Support Code
Support Code
Robot Main.vi
Dependencies
Build Specifications

8. Now deploy the code to the robot. You can run the Arcade Drive with the default FRC robot code provided. Always run the **Robot Main.vi.**

   To deploy the code to a CompactRIO device, click run ⇨ in the top left corner of the **Robot Main.vi.**


Robot Main.vi Front Panel on 2011 Robot Project.lvproj/RT CompactRIO Target
File Edit View Project Operate Tools Window Help
15pt Application Font
Enable Vision
Image Size
320x240
Finish
2011 Robot Project.lvproj/RT CompactRIO Target

The Deployment Progress Window should pop up as the code is initialized and downloaded to the CompactRIO controller. You may also see a dialog box asking if you wish to save changes to certain VIs. Should this occur, click **Save.**



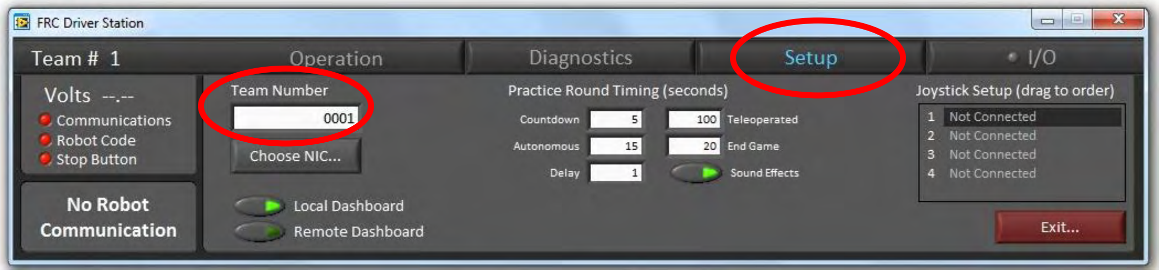
Initializing...

☐ Close on successful completion

9. Launch the FRC Driver Station.
   Before running the Arcade Drive using the default FRC robot code, you need to open the **FRC Driver Station** by **clicking** on the shortcut icon on the desktop.

The FRC Driver Station handles the communication between the computer and the robot. The Dashboard will also launch with the FRC Driver Station. Minimize the Dashboard window at this time.

10. You need to configure FRC Driver Station to communicate to your robot. Select the **Setup** tab and enter your team ID number. Team ID 0001 is shown below.



Team # 1

Volts --.--

No Robot Communication

Team Number 0001

Practice Round Timing (seconds)

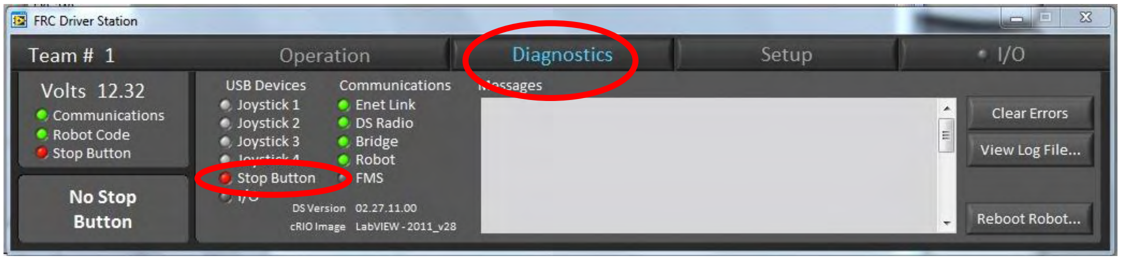
Countdown	5	100	Teleoperated
Autonomous	15	20	End Game
Delay	1		

Joystick Setup (drag to order)

1	Not Connected
2	Not Connected
3	Not Connected
4	Not Connected

11. For normal operation the FRC robot kit contains a Stop Button that is connected to the computer via USB. For the robots used in this training, there are **no USB Stop Buttons** provided. As such, we need to override this safety feature.

    Navigate to the **Diagnostics** tab and **double-click** the text “**Stop Button**” listed under USB Devices.


FRC Driver Station

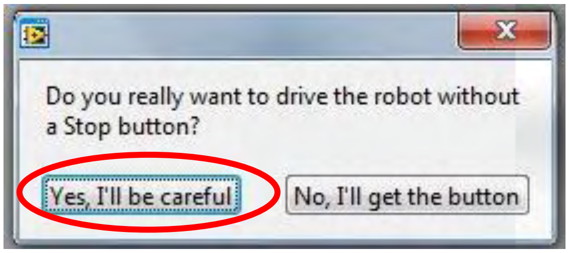
Team # 1

Volts 12.32

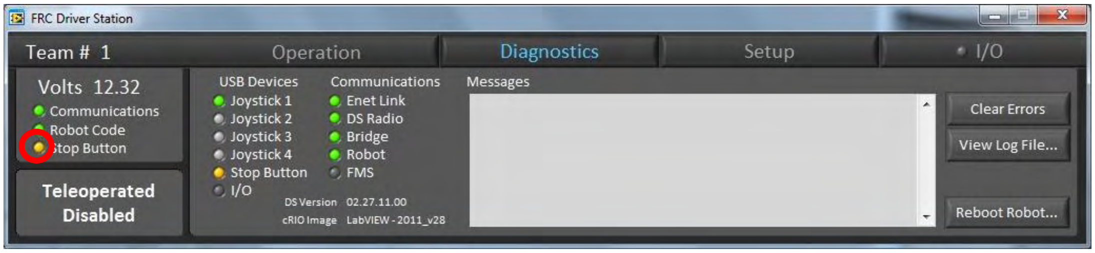
DS Version 02.27.11.00

cRIO Image LabVIEW - 2011_v28

A small dialog window will pop-up verifying the safety override changes. Select “**Yes, I’ll be careful**”.


Do you really want to drive the robot without a Stop button?

After a 20 second delay, the changes to bypass the Stop Button will have been enabled. The FRC Driver Station light associated to the Stop button on the far left should now be orange in color.


FRC Driver Station

Team # 1

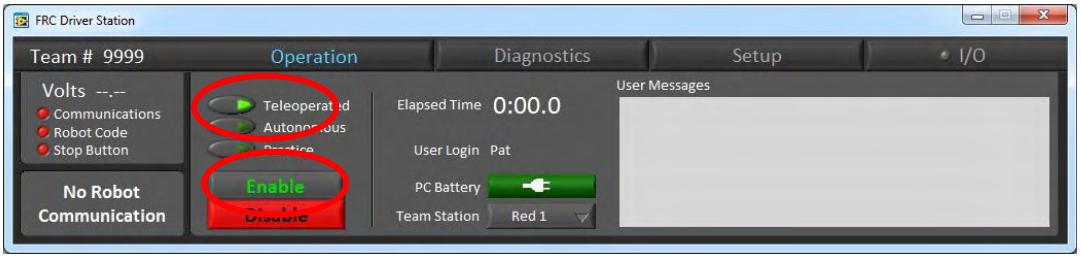
Volts 12.32

DS Version 02.27.11.00

cRIO Image LabVIEW - 2011_v28

12. When the lights of the Communications and Robot Code turn green, and the light of the Stop Button turns orange (or green if a USB Stop Button is connected) the robot is ready to run.

13. From the Operation tab select **Teleoperated** and then click **Enable** to start the robot.


FRC Driver Station

Team # 9999

Volts --.--

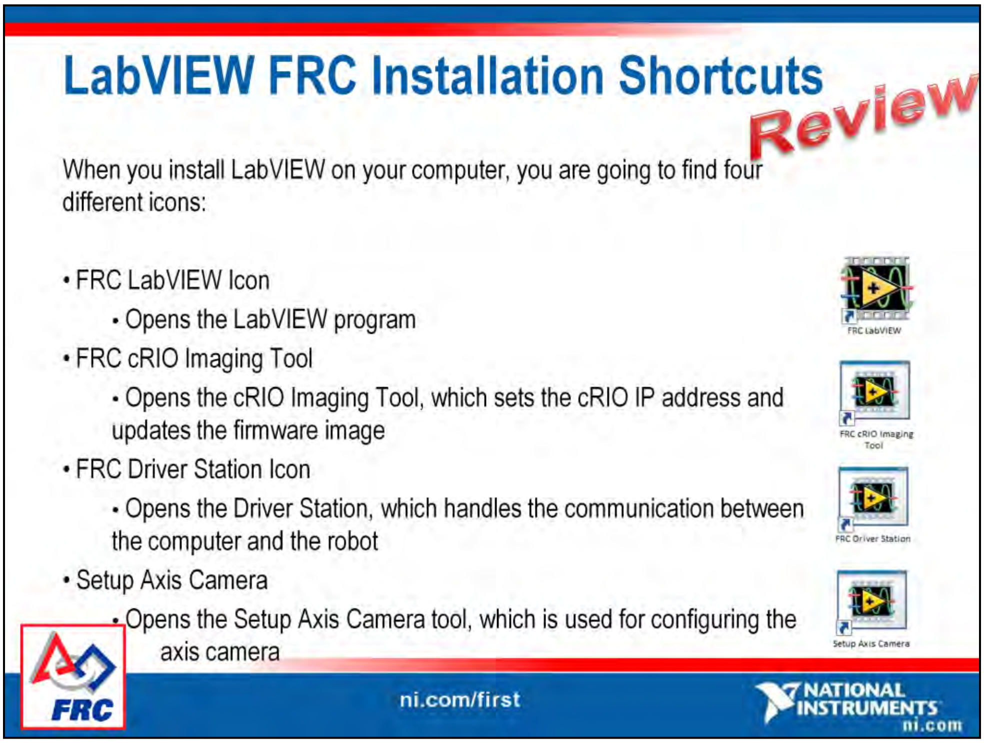
Elapsed Time 0:00.0

User Login Pat

Team Station Red 1

14. Now you can drive the robot with the joystick.

END OF FRC EXERCISE 1



LabVIEW FRC Installation Shortcuts

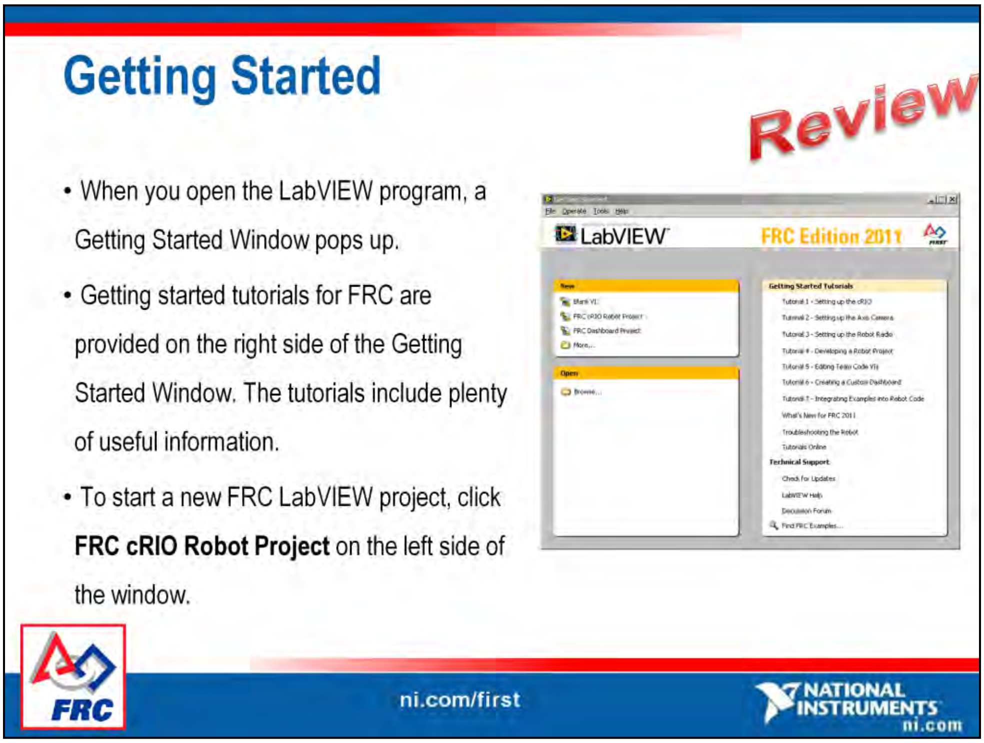
Review

When you install LabVIEW on your computer, you are going to find four different icons:

- FRC LabVIEW Icon
  - Opens the LabVIEW program
- FRC cRIO Imaging Tool
  - Opens the cRIO Imaging Tool, which sets the cRIO IP address and updates the firmware image
- FRC Driver Station Icon
  - Opens the Driver Station, which handles the communication between the computer and the robot
- Setup Axis Camera
  - Opens the Setup Axis Camera tool, which is used for configuring the axis camera

ni.com/first

Review of LabVIEW FRC Robot Exercise 1



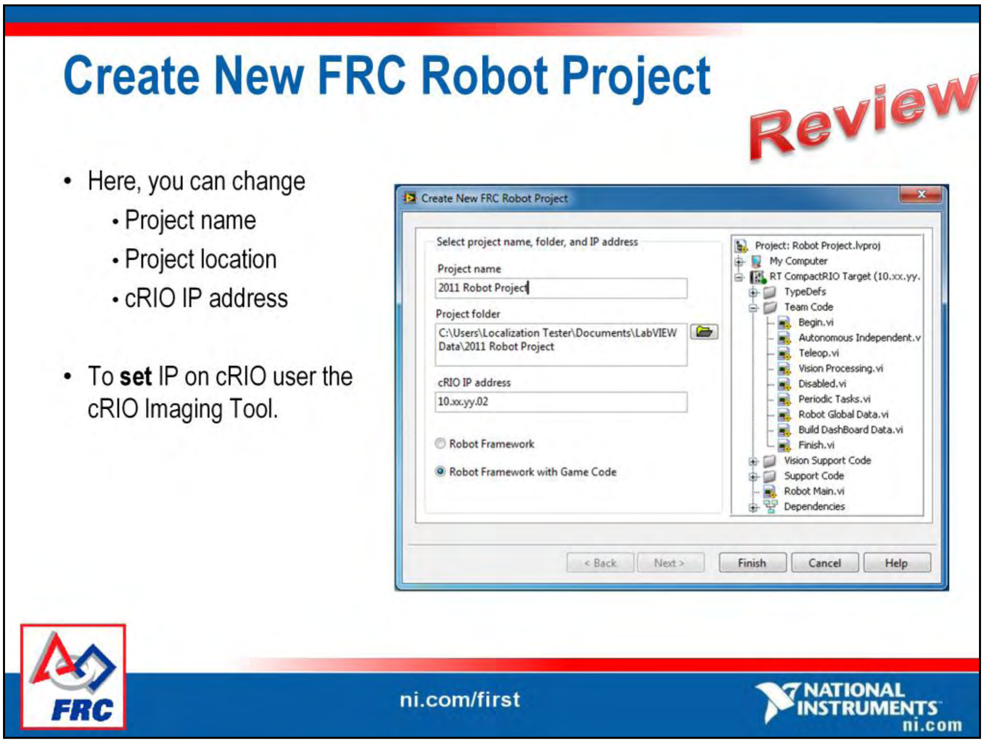
Getting Started

Review

- When you open the LabVIEW program, a Getting Started Window pops up.
- Getting started tutorials for FRC are provided on the right side of the Getting Started Window. The tutorials include plenty of useful information.
- To start a new FRC LabVIEW project, click **FRC cRIO Robot Project** on the left side of the window.

File Operate Tools Help
LabVIEW
FRC Edition 2011
New
Blank VI
FRC cRIO Robot Project
FRC Dashboard Project
More...
Open
Browse...
Getting Started Tutorials
Tutorial 1 - Setting up the cRIO
Tutorial 2 - Setting up the Axis Camera
Tutorial 3 - Setting up the Robot Radio
Tutorial 4 - Developing a Robot Project
Tutorial 5 - Editing Team Code VIs
Tutorial 6 - Creating a Custom Dashboard
Tutorial 7 - Integrating Examples into Robot Code
What's New for FRC 2011
Troubleshooting the Robot
Tutorials Online
Technical Support
Check for Updates
LabVIEW Help
Discussion Forum
Find FRC Examples...
FRC
ni.com/first
NATIONAL INSTRUMENTS
ni.com

Review of LabVIEW FRC Robot Exercise 1



Create New FRC Robot Project

Review

- Here, you can change
  - Project name
  - Project location
  - cRIO IP address
- To **set** IP on cRIO user the cRIO Imaging Tool.

Create New FRC Robot Project

Select project name, folder, and IP address

Project name: 2011 Robot Project

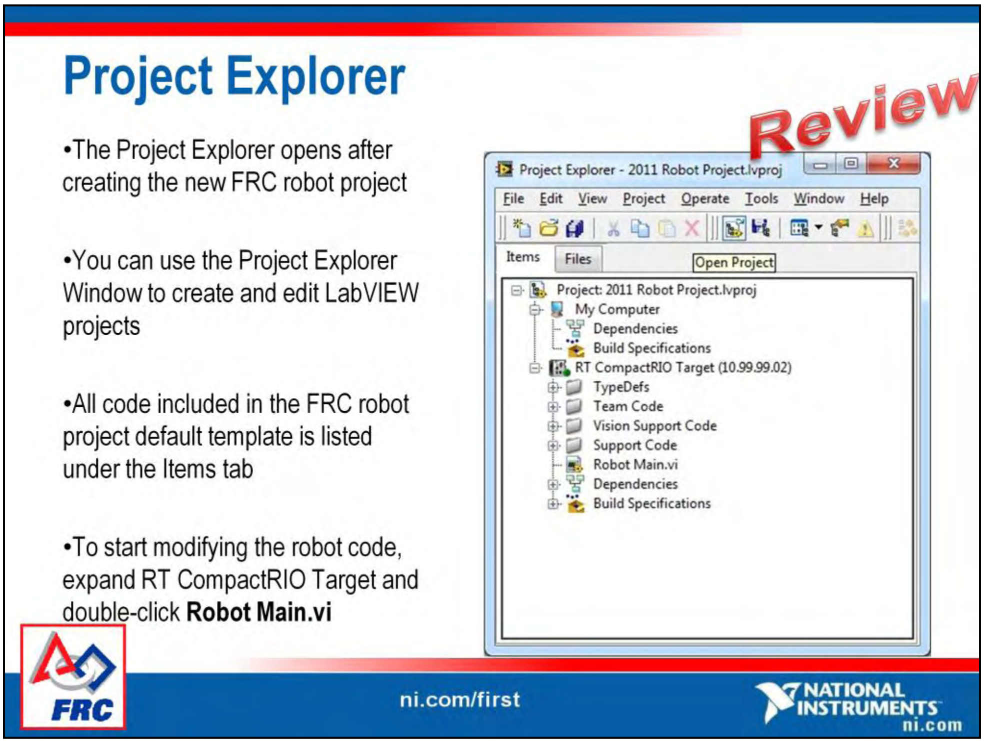
Project folder: C:\Users\Localization Tester\Documents\LabVIEW Data\2011 Robot Project

cRIO IP address: 10.xx.yy.02

○ Robot Framework

● Robot Framework with Game Code

Review of LabVIEW FRC Robot Exercise 1



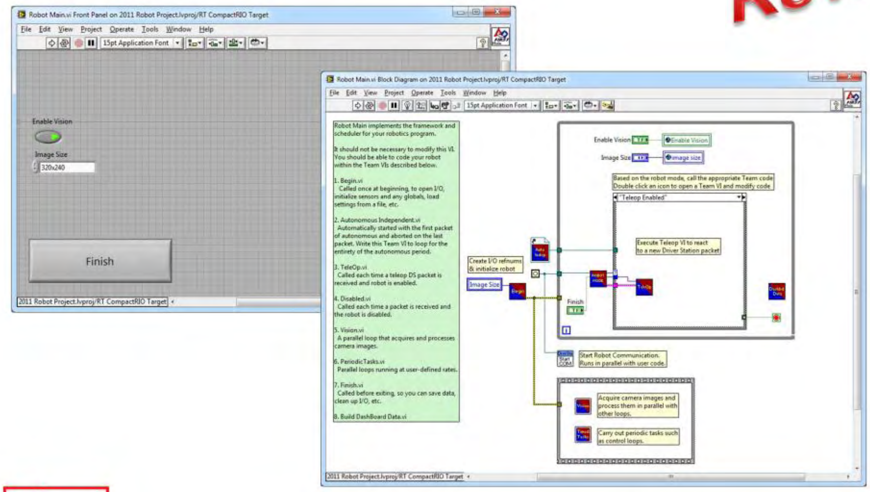
Project Explorer

- The Project Explorer opens after creating the new FRC robot project
- You can use the Project Explorer Window to create and edit LabVIEW projects
- All code included in the FRC robot project default template is listed under the Items tab
- To start modifying the robot code, expand RT CompactRIO Target and double-click **Robot Main.vi**

Review of LabVIEW FRC Robot Exercise 1

Robot Main.VI

Review


Robot Main.vi Front Panel on 2011 Robot Project.lvproj/RT CompactRIO Target
File Edit View Project Operate Tools Window Help
15pt Application Font
Enable Vision
Image Size
320x240
Finish
2011 Robot Project.lvproj/RT CompactRIO Target
Robot Main.vi Block Diagram on 2011 Robot Project.lvproj/RT CompactRIO Target
File Edit View Project Operate Tools Window Help
15pt Application Font
Enable Vision
Image Size
Enable Vision
image size
"Teleop Enabled"
Image Size
Begin
Finish
TeleOp
2011 Robot Project.lvproj/RT CompactRIO Target

Robot Main implements the framework and scheduler for your robotics program.

It should not be necessary to modify this VI. You should be able to code your robot within the Team VIs described below.

1. Begin.vi
   Called once at beginning, to open I/O, initialize sensors and any globals, load settings from a file, etc.

2. Autonomous Independent.vi
   Automatically started with the first packet of autonomous and aborted on the last packet. Write this Team VI to loop for the entirety of the autonomous period.

3. TeleOp.vi
   Called each time a teleop DS packet is received and robot is enabled.

4. Disabled.vi
   Called each time a packet is received and the robot is disabled.

5. Vision.vi
   A parallel loop that acquires and processes camera images.

6. PeriodicTasks.vi
   Parallel loops running at user-defined rates.

7. Finish.vi
   Called before exiting, so you can save data, clean up I/O, etc.

8. Build DashBoard Data.vi

Based on the robot mode, call the appropriate Team code
Double click an icon to open a Team VI and modify code

Execute Teleop VI to react to a new Driver Station packet

Create I/O refnums & initialize robot

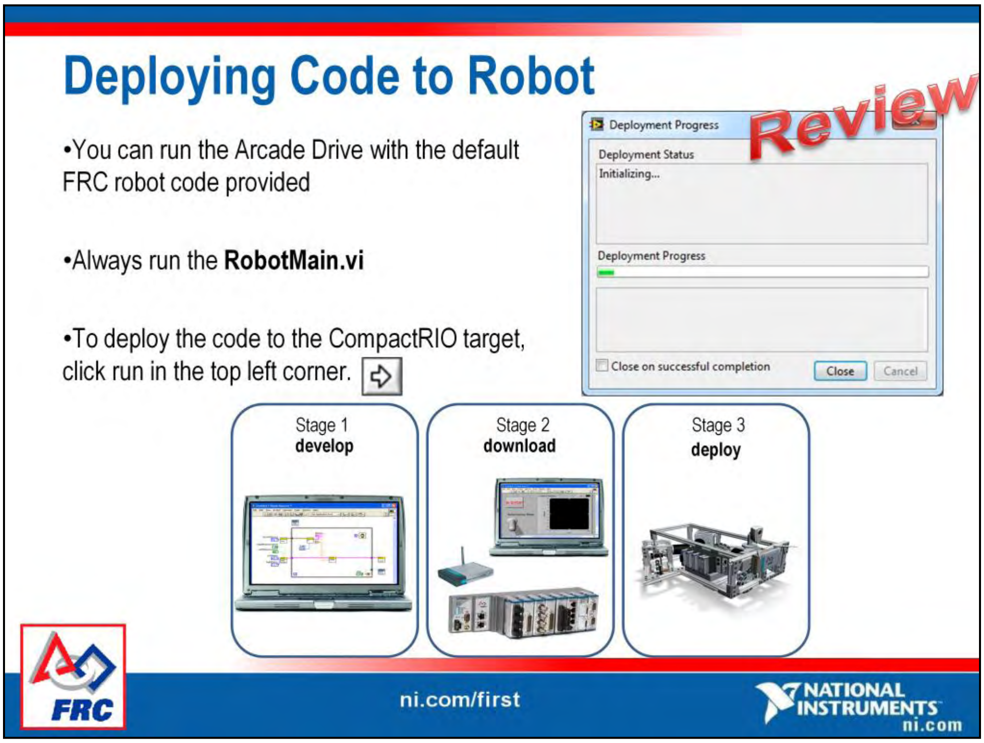
Start Robot Communication. Runs in parallel with user code.

Acquire camera images and process them in parallel with other loops.

Carry out periodic tasks such as control loops.

ni.com/first

Review of LabVIEW FRC Robot Exercise 1



Deploying Code to Robot

- •You can run the Arcade Drive with the default FRC robot code provided
- •Always run the **RobotMain.vi**
- •To deploy the code to the CompactRIO target, click run in the top left corner.

Deployment Progress

Deployment Status

Initializing...

Deployment Progress

□ Close on successful completion

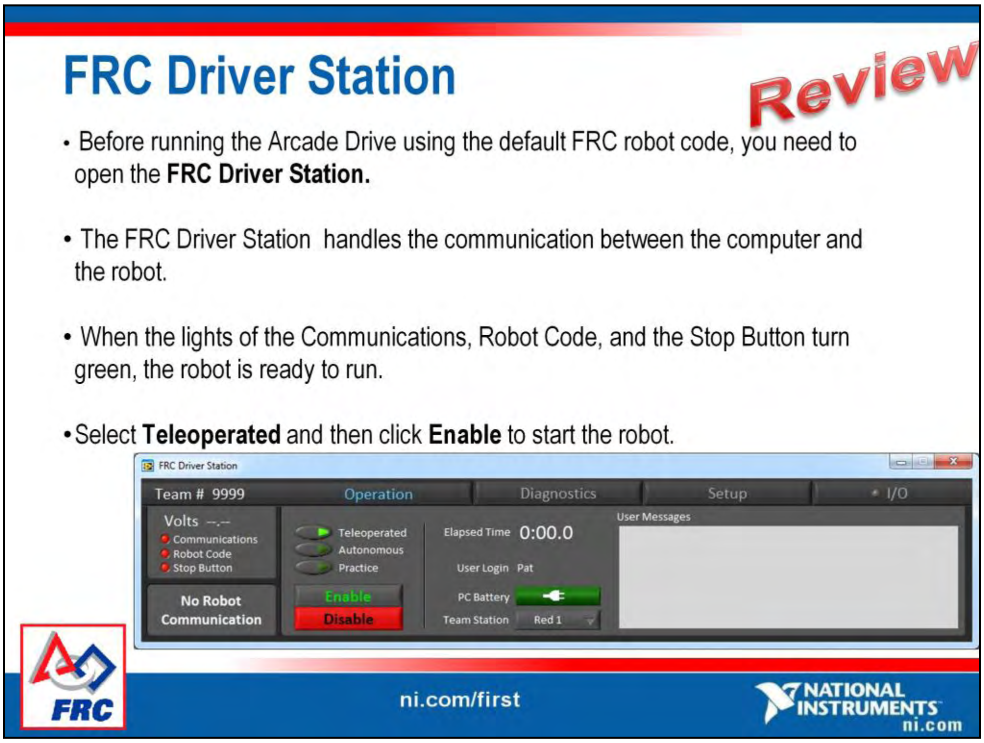
Stage 1 **develop**

Stage 2 **download**

Stage 3 **deploy**

ni.com/first

Review of LabVIEW FRC Robot Exercise 1



FRC Driver Station

Review

- Before running the Arcade Drive using the default FRC robot code, you need to open the **FRC Driver Station.**
- The FRC Driver Station handles the communication between the computer and the robot.
- When the lights of the Communications, Robot Code, and the Stop Button turn green, the robot is ready to run.
- Select **Teleoperated** and then click **Enable** to start the robot.

FRC Driver Station

Team # 9999

Volts --.--

Communications
Robot Code
Stop Button

No Robot Communication

Elapsed Time 0:00.0

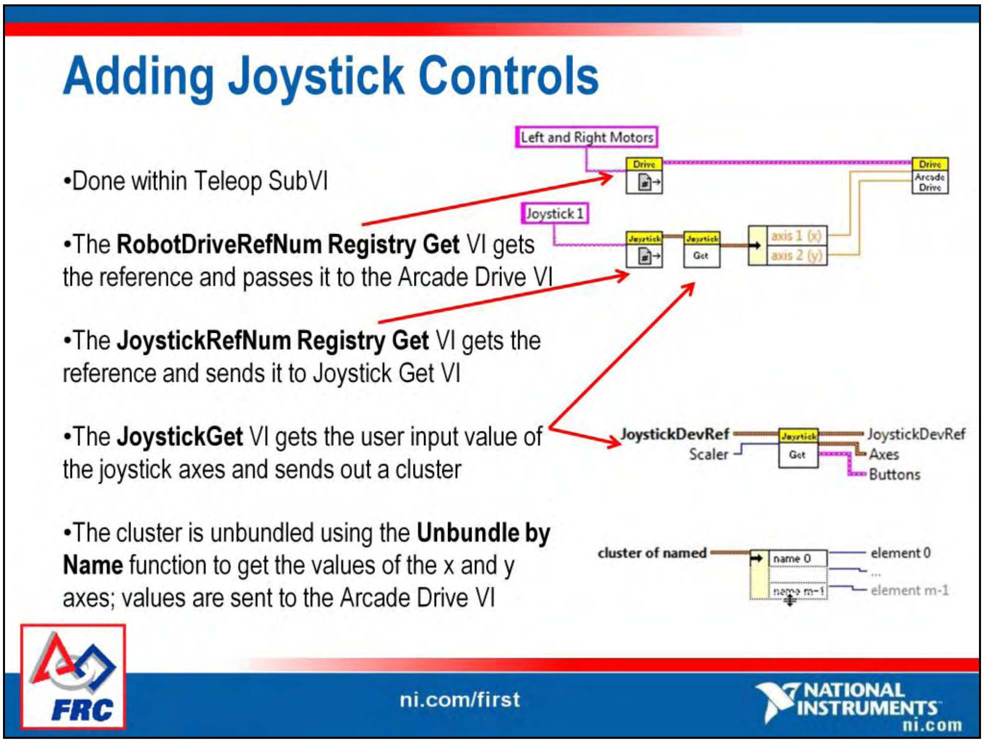
User Login Pat

Team Station Red 1

User Messages

ni.com/first

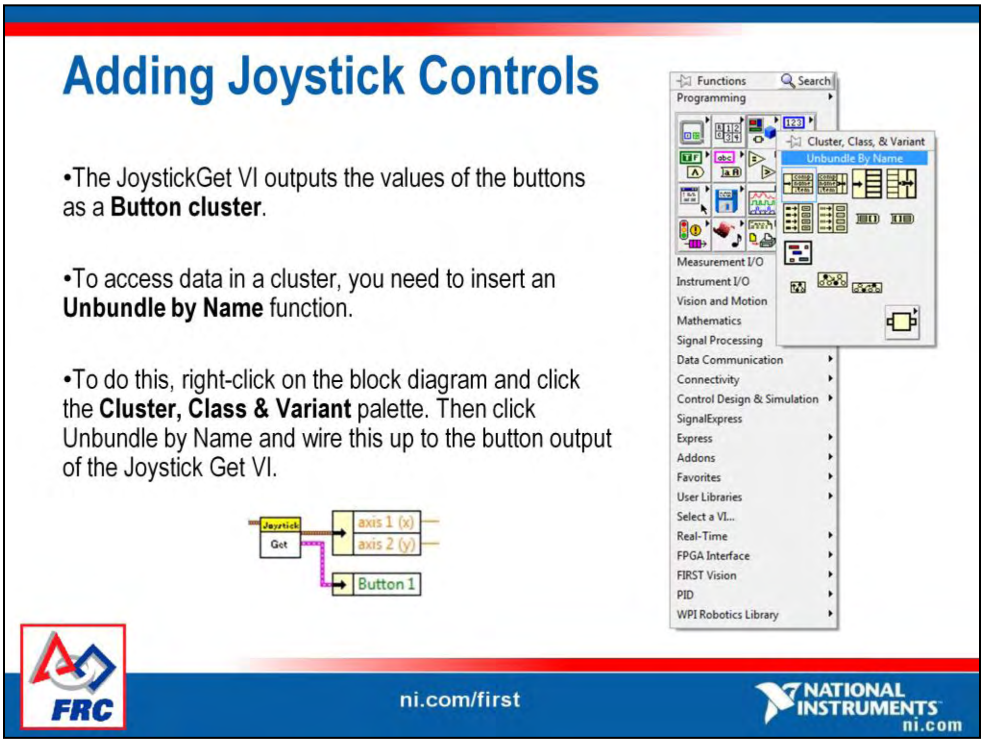
Review of LabVIEW FRC Robot Exercise 1



Adding Joystick Controls

- Done within Teleop SubVI
- The **RobotDriveRefNum Registry Get** VI gets the reference and passes it to the Arcade Drive VI
- The **JoystickRefNum Registry Get** VI gets the reference and sends it to Joystick Get VI
- The **JoystickGet** VI gets the user input value of the joystick axes and sends out a cluster
- The cluster is unbundled using the **Unbundle by Name** function to get the values of the x and y axes; values are sent to the Arcade Drive VI

ni.com/first



Adding Joystick Controls

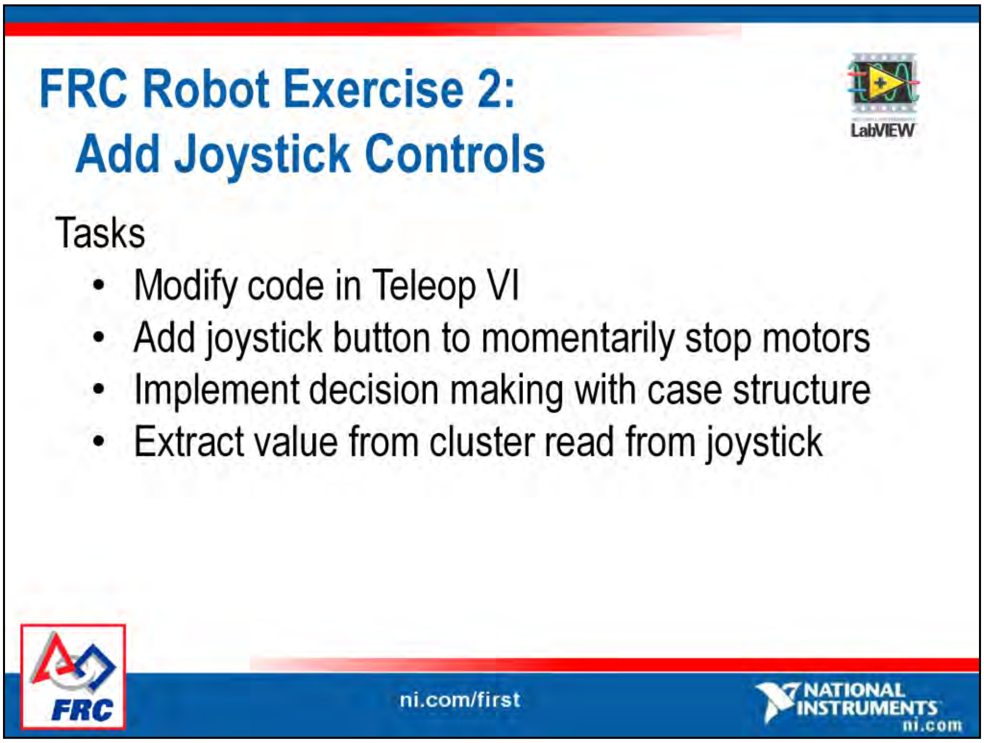
- The JoystickGet VI outputs the values of the buttons as a **Button cluster**.
- To access data in a cluster, you need to insert an **Unbundle by Name** function.
- To do this, right-click on the block diagram and click the **Cluster, Class & Variant** palette. Then click Unbundle by Name and wire this up to the button output of the Joystick Get VI.

ni.com/first

Review: Key LabVIEW Structures for *FIRST*—From Introduction to NI LabVIEW Section

The JoystickGet VI outputs the values of the buttons as a **Button cluster**. To access data in a cluster, you need to insert an **Unbundle by Name** Function. To do this, right-click on the block diagram and click the **Cluster, Class & Variant** palette. Then click Unbundle by Name. Wire this up to the button output of the Joystick Get.

Button 1 now appears in the Unbundle by Name function. Note the color of Button 1 is green because it is a Boolean value. To add a button, simply drag the bottom of the Unbundle by Name function to expand it. To change the button, simply click and select the desired button.



FRC Robot Exercise 2: Add Joystick Controls

Tasks

- Modify code in Teleop VI
- Add joystick button to momentarily stop motors
- Implement decision making with case structure
- Extract value from cluster read from joystick

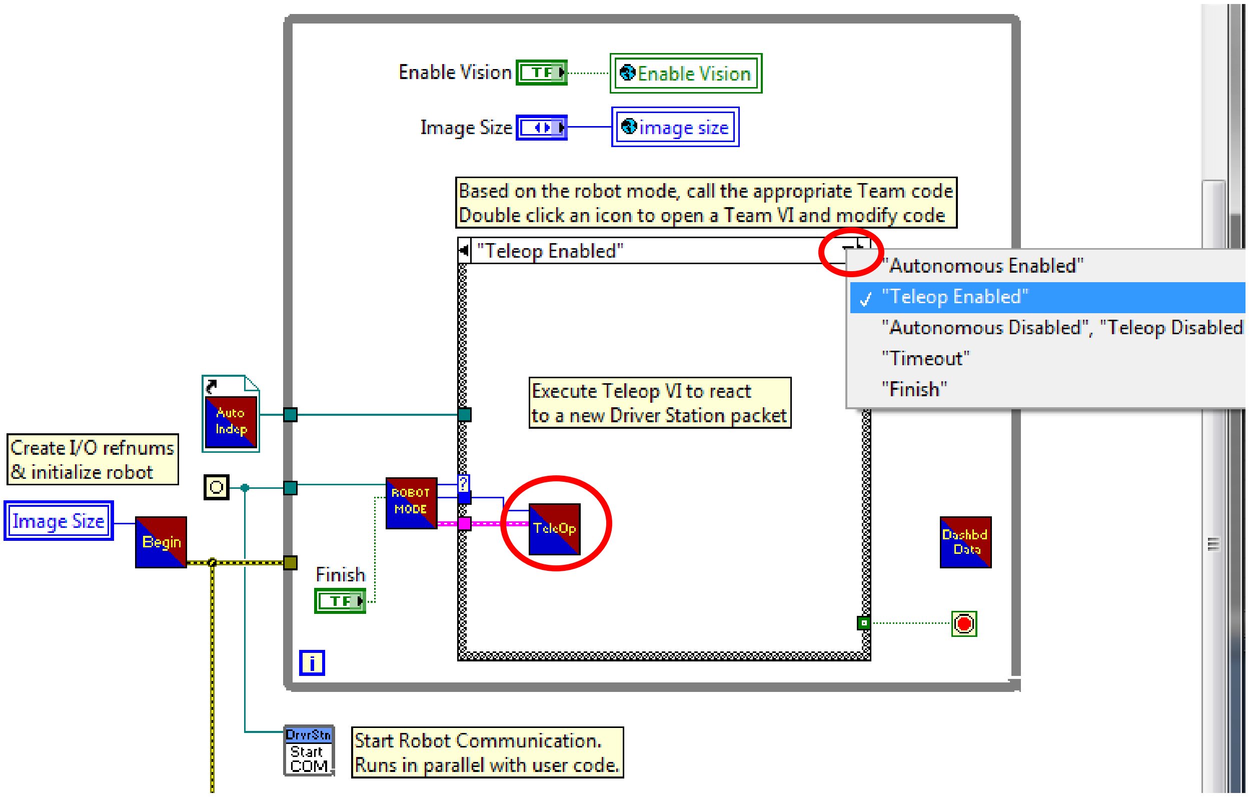
LabVIEW
FRC
ni.com/first
NATIONAL INSTRUMENTS
ni.com

FRC Robot Exercise 2: Add Joystick Controls

In this exercise, modify code in the Teleop SubVI and add a joystick button to momentarily stop motors while driving. This functionality will act like a brake in our robot code. Use clusters and a case structure to implement this code.

Note: Normally when adding a new device, you need to open and close the references to this device in Begin.VI and Finish.VI. This is shown in the following exercises. In this case, however, the joystick has already been added to the code template by default, so you need to modify only the code in the Teleop.VI.

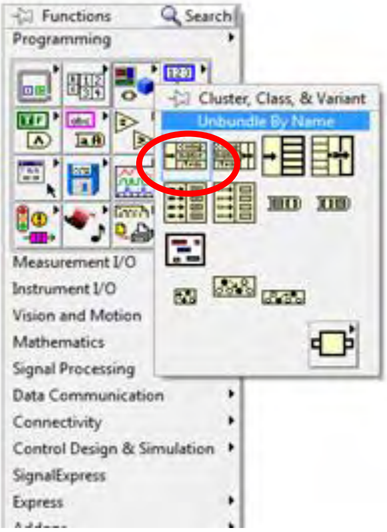
1. **Open** the **block diagram of Teleop.VI**. To do this, select the "Teleop Enabled" case from the case structure on the Robot Main.VI block diagram. Then double-click on the subVI named Teleop.VI. If Robot Main is not open, you can also double-click Teleop.VI from the Project Explorer Window. Once the VI is launched, switch to the block diagram by selecting <Ctrl-E>.


Enable Vision
Enable Vision
Image Size
image size
Based on the robot mode, call the appropriate Team code
Double click an icon to open a Team VI and modify code
"Teleop Enabled"
"Autonomous Enabled"
"Teleop Enabled"
"Autonomous Disabled", "Teleop Disabled
"Timeout"
"Finish"
Auto Indep
Create I/O refnums & initialize robot
Image Size
Begin
ROBOT MODE
Execute Teleop VI to react to a new Driver Station packet
TeleOp
Dashbd Data
Finish
DrvrStn Start COM
Start Robot Communication. Runs in parallel with user code.

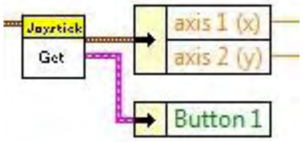
The Joystick Get VI outputs the values of the buttons as a **Button cluster**.

To access data in a cluster, you need to insert an **Unbundle by Name** Function.

2. To do this, right-click on the block diagram and from the Functions palette under Programming click the **Cluster, Class & Variant** palette. Then click Unbundle by Name.

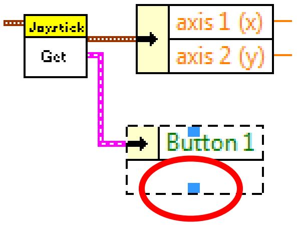


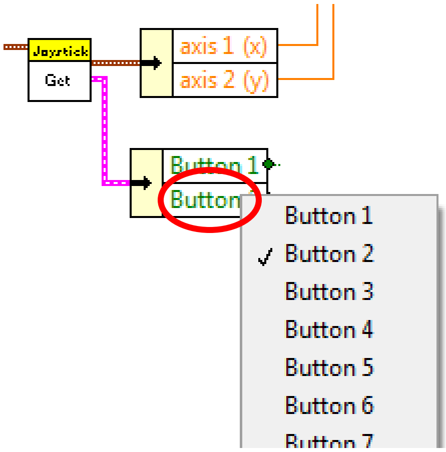
3. Wire this to the button output of the Joystick Get. If you are unsure where to wire, open the Context Help (Ctrl-H) and hover the mouse over the VI. This shows where you can make all of the connections. Button 1 is the “trigger” on the back of the joystick.



Button 1 now appears in the Unbundle by Name function. Note the color of Button 1 is green because it is a Boolean (True/False) value.

NOTE: To add another button, simply drag the bottom of the Unbundle by Name function to expand it. To change the button, left-click and select the desired button. For this exercise only one button is needed.

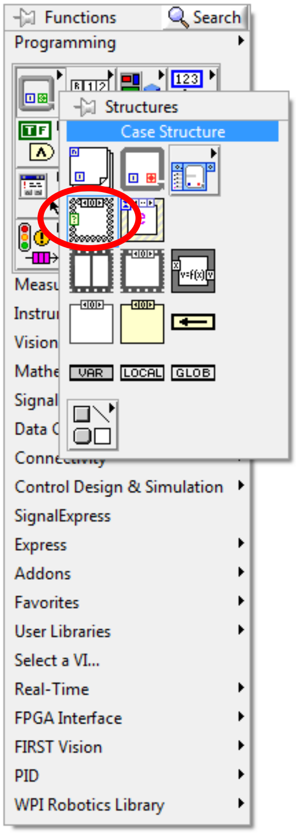




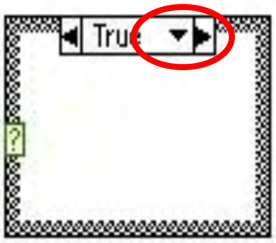
To perform an action based on the value of the button, you need a **case structure**. A case structure contains one or more subdiagrams, or cases. The value wired to the selector terminal determines which case to execute.

The value can be a string, integer, or enumerated type, but it is most often a **Boolean**.

4. To insert a case structure, right-click on the block diagram, click **Structure** palette, and then click Case Structure. Click and drag on the block diagram to create the case structure.



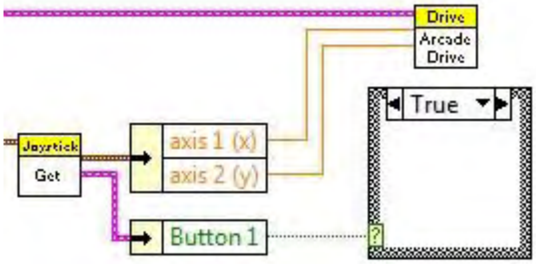
A default case structure has two cases: True and False. To change between subdiagrams, click the arrows on the top of the case structure or select it from the pull-down menu.



A button has two states: **True for pressed** and **False for not pressed**.

For a momentary stop button, the motors should stop when Button 1 is true and keep running when Button 1 is false.

5. To do this in LabVIEW, wire the output of Button 1 from the Unbundle by Name Function to the case selector on the left edge of the case structure.

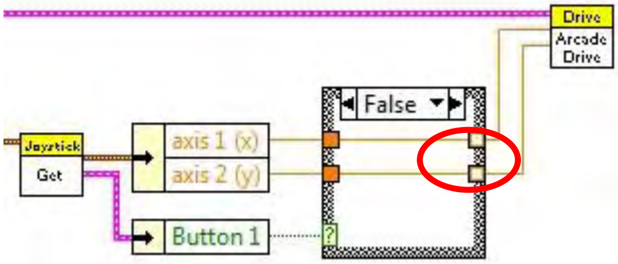


This allows the case structure to execute different code based on the Boolean value of Button 1.

In the False Case (when Button 1 is not pressed), the motors should keep running.

Therefore, the x- and y-axis values should go through the case structure without any change.

6. To wire the false case, delete the x- and y-axis wires and rewire them to Arcade Drive **through the case structure**. Note that the wires went through only the one visible case.



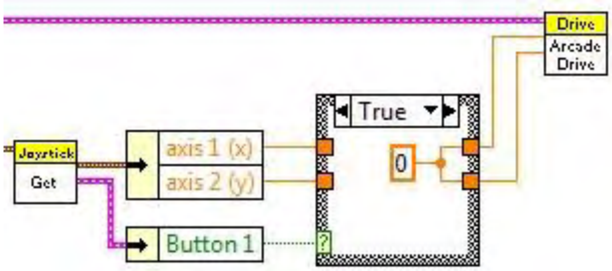
Four blocks are created when you are wiring through the case structure. The two blocks on the left of the case structure are where the wires go in and are called the **input tunnels**.

The two blocks on the right of the case structure are where the wires come out and are called the **output tunnels**.

In the True Case, when Button 1 is pressed, the motors should stop. Therefore, the x- and y-axis values that we send to the Robot Drive function should be zero.

To do this, change to the True Case and wire a zero constant to the output tunnels.

7. To insert a constant, right-click on one of the output tunnels, click **Create**, and then click **Constant**. Note that you can reuse the constant for the second output tunnel.



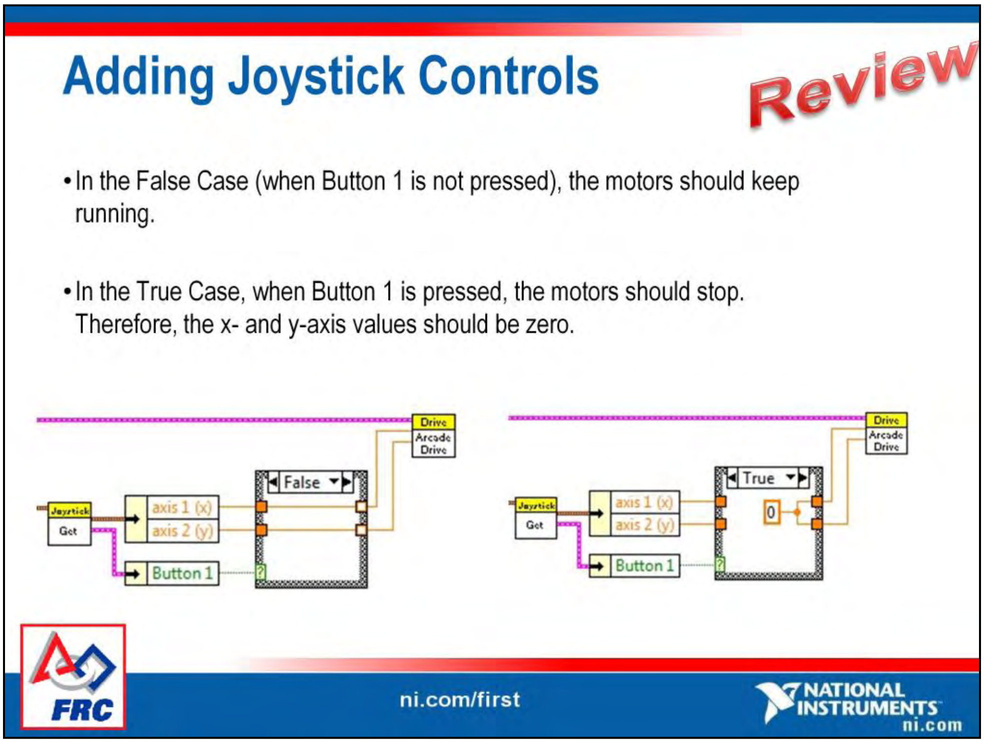
A momentary motor brake has now been created.

8. Close and save Teleop VI.

9. Return to Robot Main and deploy the code to test it out by click the Run button.

NOTE: Remember the FRC Driver Station needs to be running.

END OF FRC EXERCISE 2



Adding Joystick Controls

Review

- In the False Case (when Button 1 is not pressed), the motors should keep running.
- In the True Case, when Button 1 is pressed, the motors should stop. Therefore, the x- and y-axis values should be zero.

ni.com/first

NATIONAL INSTRUMENTS ni.com

To access data in a cluster, you need to insert an **Unbundle by Name** function.

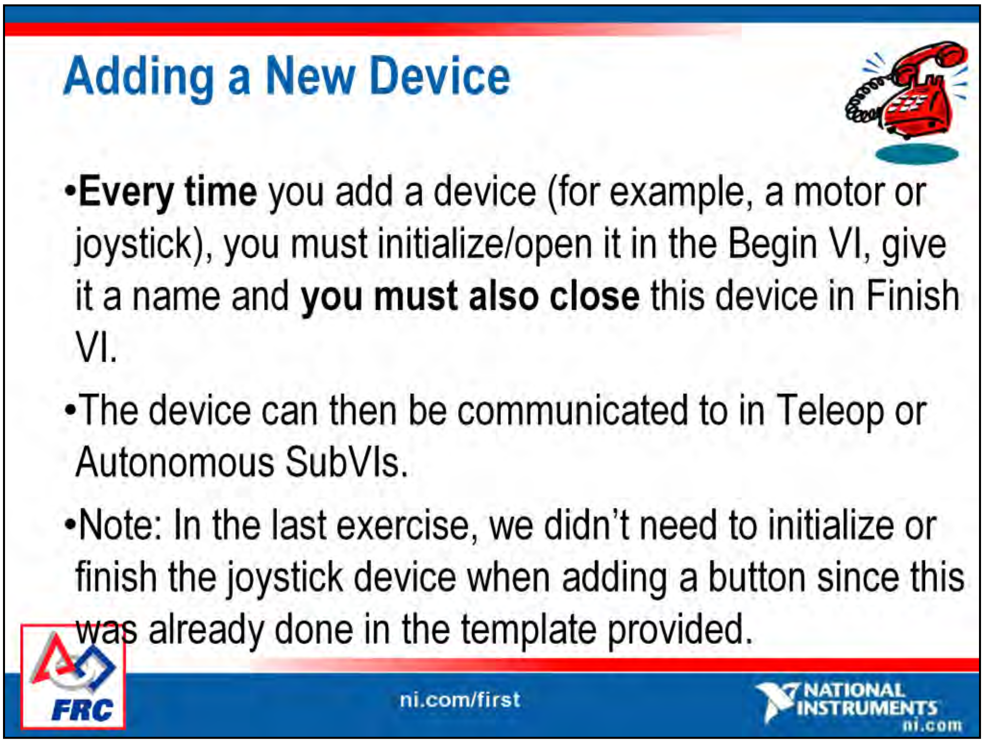
A **Case Structure** contains one or more subdiagrams, or cases. The value wired to the selector terminal determines which case to execute.

For a momentary stop button, remember that the motors should stop when Button 1 is true and keep running when Button 1 is false. Therefore, the x- and y-axis values should go through the case structure without any change. To wire the false case, delete the x- and y-axis wires and rewire them to Arcade Drive **through the case structure**. Note that the wires only went through the one visible case.

Four blocks are created when you are wiring through the case structure. The two blocks on the left of the case structure are where the wires go in and are called the **input tunnels**.

In the True Case, when Button 1 is pressed, the motors should stop. Therefore, the x- and y-axis values should be zero.

To do this, change to the true case and wire a zero constant to the output tunnels. To insert a constant, right-click on one of the output tunnels, click **Create**, and then click **Constant**. Note that you can reuse the constant for the second output tunnel.



Adding a New Device

- **Every time** you add a device (for example, a motor or joystick), you must initialize/open it in the Begin VI, give it a name and **you must also close** this device in Finish VI.
- The device can then be communicated to in Teleop or Autonomous SubVIs.
- Note: In the last exercise, we didn't need to initialize or finish the joystick device when adding a button since this was already done in the template provided.

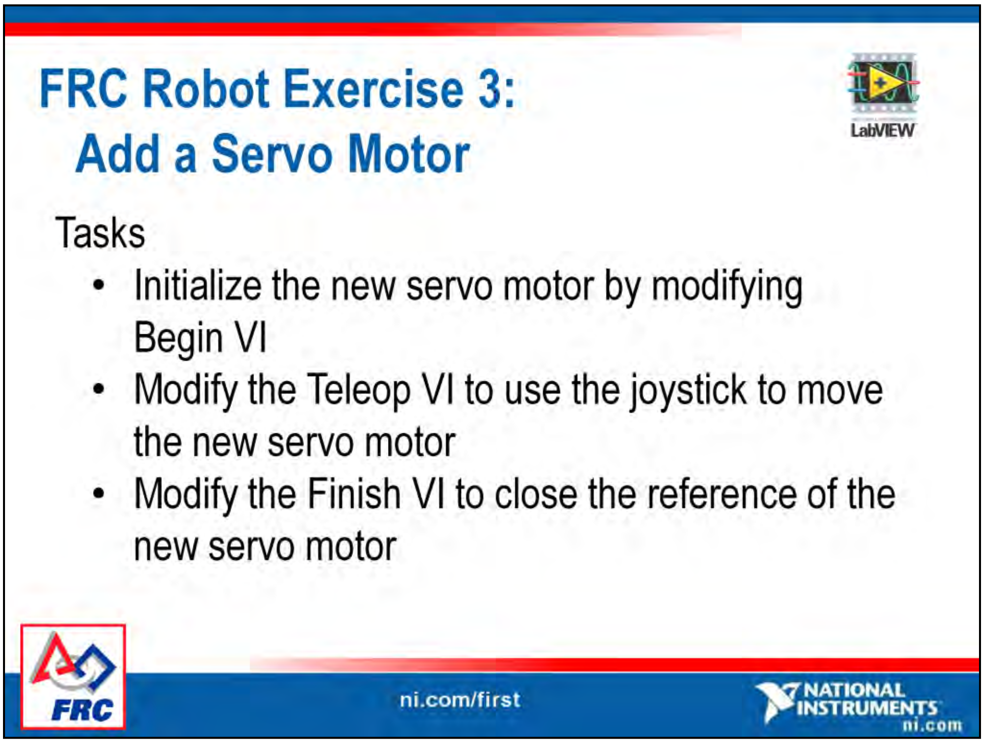
ni.com/first

Think of this like a telephone conversation.

First you need to dial the phone number and say whom you wish to speak with. This is the initialize stage in the Begin.VI.

Next you talk back and forth, back and forth. This is done in the Teleop.VI (or Autonomous.VI) as the device (joystick) communicates to another device (such as the motors). This occurs over and over again as this code is called in a loop.

Finally, you hang up and end the call. This is the close resources and clean-up stage in the Finish.VI.



FRC Robot Exercise 3: Add a Servo Motor

LabVIEW

Tasks

- Initialize the new servo motor by modifying Begin VI
- Modify the Teleop VI to use the joystick to move the new servo motor
- Modify the Finish VI to close the reference of the new servo motor

FRC
ni.com/first
NATIONAL INSTRUMENTS
ni.com

FRC Robot Exercise 3: Add a Servo Motor

In this exercise, add a new servo motor device to the project. This is accomplished by initializing the new servo motor by modifying Begin SubVI, modifying the Teleop SubVI to use the joystick to move the new servo motor, and finally modifying the Finish SubVI to close the reference of the new servo motor.

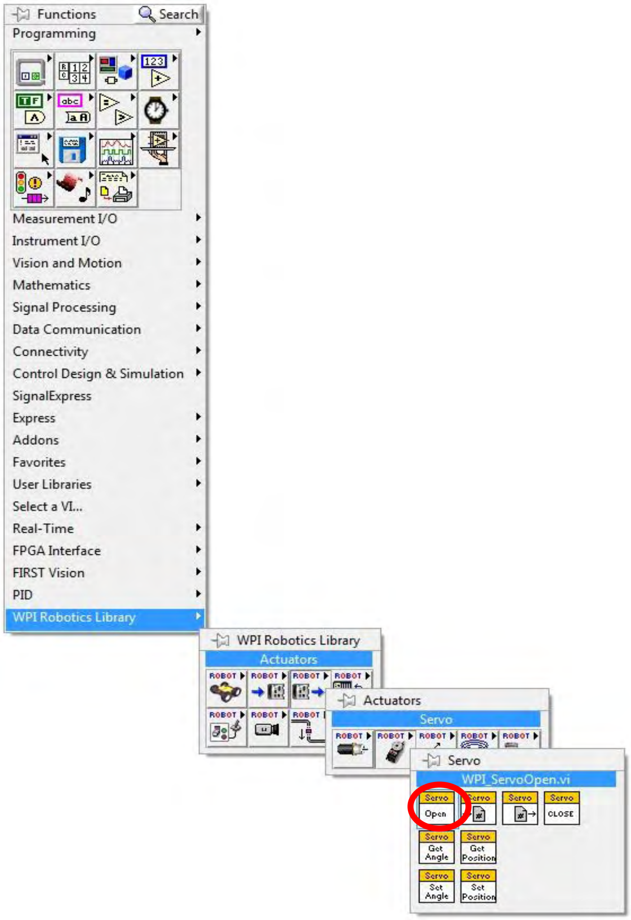
Servo motors are often used to control robot arms and clamps.

Servo Initialization

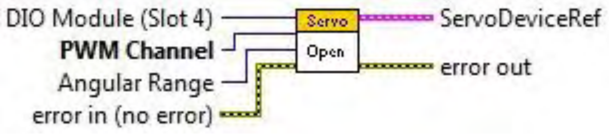
1. Open the block diagram of the Begin.vi. Find the Begin.vi in the Project Explorer for the FRC project created in Exercise 1 or in the block diagram of Robot Main.vi.

2. Use the **ServoOpen VI** to configure the servo.

   To insert it, right-click on the block diagram, click **WPI Robotics Library**, then click **Actuators** palette, and then click **Servo** palette.

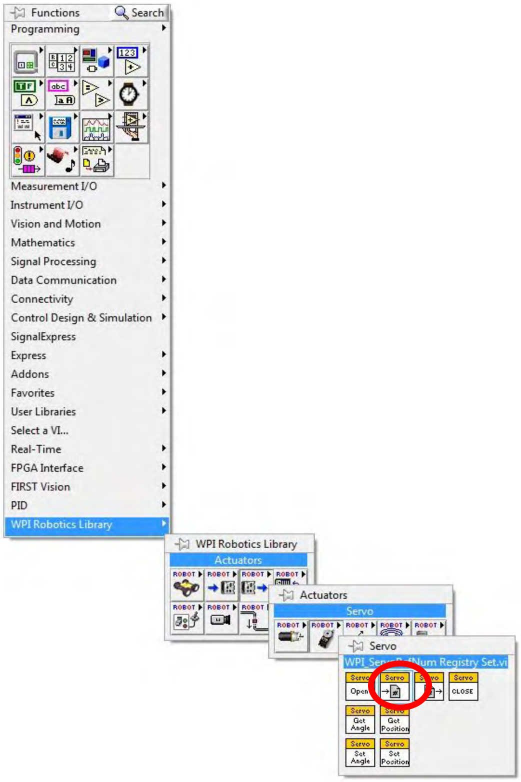


3. Select the **ServoOpen VI** and place it on the block diagram.

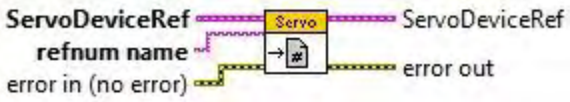


You also need the **Servo Reference Registry Set** VI to register the servo. (May be called WPI_ServoRefNum Registry Set.vi.)

4. To insert it, right-click on the block diagram, click **WPI Robotics Library**, then click **Actuators** palette, and then click **Servo** palette.



5. Select the **Servo Reference Registry Set** VI and place it on the block diagram.

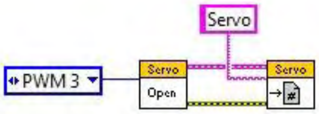


6. Create a PWM constant that corresponds to the channel where the servo is connected.

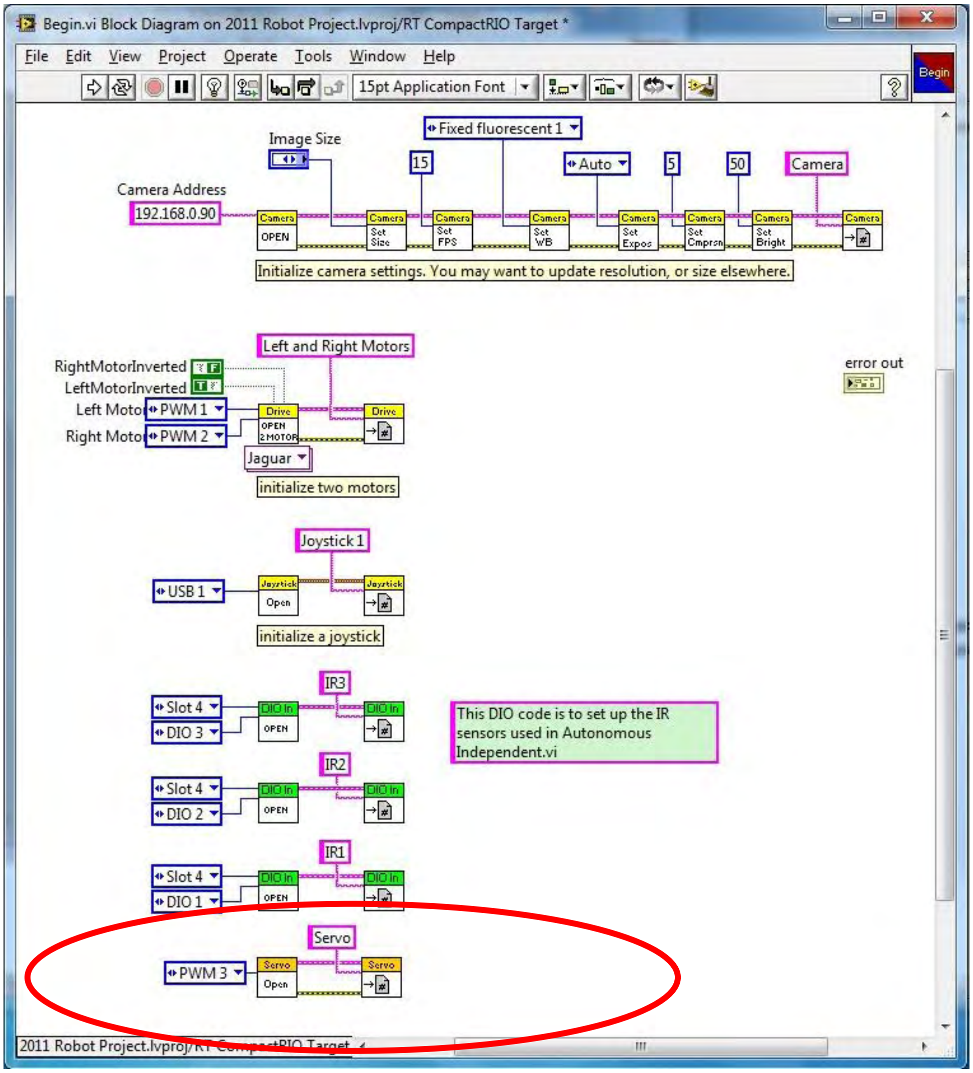
   To do this, right-click on the **PWM Channel** input on the **ServoOpen** VI, click **Create**, and then click **Constant**. Select the desired channel (PWM3) from the pull-down menu of the PWM constant.

7. Wire the reference and error cluster between the **ServoOpen** VI and the **Servo Reference Registry Set** VI as shown below.

8. Name the servo reference “Servo.” To do this, create a string constant wired into the **Refnum Name** Input of the Servo Reference Registry Set VI. Do so by right-clicking the Refnum Name input terminal and selecting create»constant.



The block diagram to Begin.VI should resemble the image below, with the added code circled. The labels under the VIs may not appear depending on your LabVIEW settings.



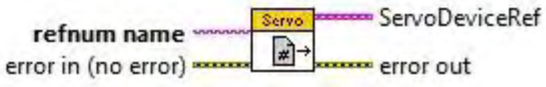
9. Close and save Begin.VI.

Servo Reference Close

Whenever a device is opened in the Begin.vi, it should also be closed in the Finish.vi.

10. Open the block diagram of the **Finish.vi.** You can find the Finish.vi in the Project Explorer for the FRC project created in Exercise 1 or in the "Finish" case of the case structure from the block diagram of Robot Main.vi.

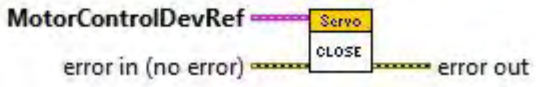
11. From the Servo palette, insert a **ServoRefNum Registry Get** VI to get the reference of the previously configured servo. Note that you can find the palette in the same location as steps 2 and 4.



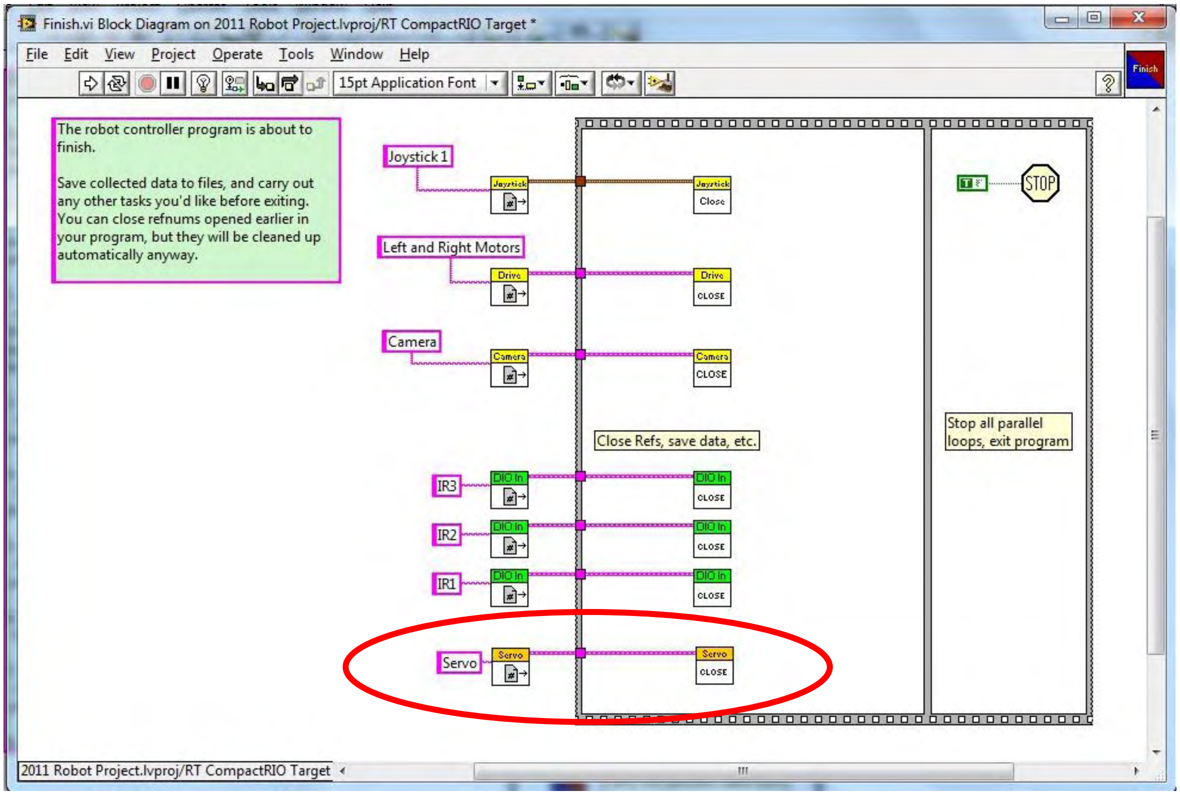
HINT: If you need more room on the Finish.vi block diagram, try holding down the Ctrl key on the keyboard while selecting a portion of the screen with the mouse. This "pushes out" the block diagram relative to the size of the box you created with the mouse.

12. Create a string constant on the **Refnum Name** input and type in the name of the device. This was "Servo" or the name you used in step 8.

13. Place a **ServoClose** VI to close the reference of the servo.



14. Wire the reference cluster from the Servo Reference Get VI to the Servo Reference Close VI. The Finish.VI block diagram should resemble the image below, with the added code circled. The labels under the VIs may not appear depending on your LabVIEW settings.



Finish.vi Block Diagram on 2011 Robot Project.lvproj/RT CompactRIO Target *

The robot controller program is about to finish.

Save collected data to files, and carry out any other tasks you'd like before exiting. You can close refnums opened earlier in your program, but they will be cleaned up automatically anyway.

Close Refs, save data, etc.

Stop all parallel loops, exit program

2011 Robot Project.lvproj/RT CompactRIO Target

15. Close and save **Finish VI**.

Move Servo

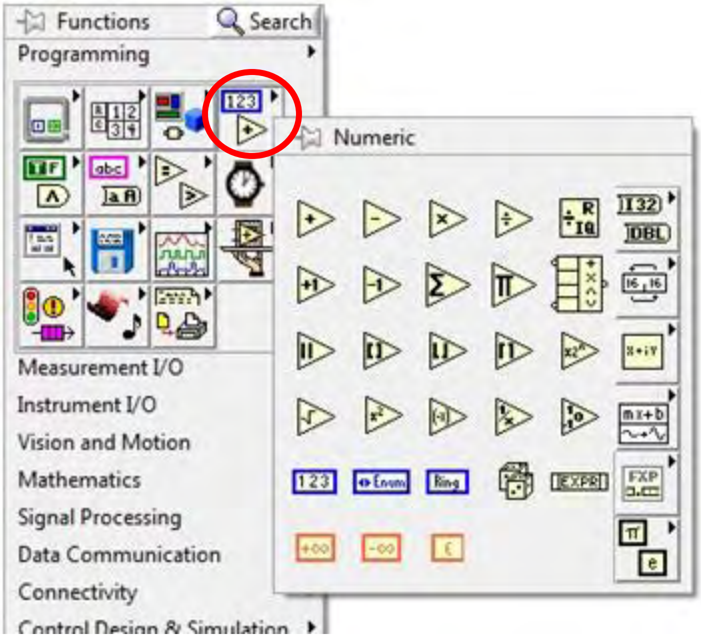
16. Open the Teleop SubVI to modify the code. You can find Teleop.vi in the Project Explorer for the FRC project created in Exercise 1 or in the “Teleop Enabled” case of the case structure from the block diagram of Robot Main.vi.

First you need to convert the throttle axis value to a servo angle. The throttle is the round control at the base of the joystick between buttons 8 and 9.

The value of the throttle ranges from -1 to 1, and the range of the servo angle ranges from 0 to 170 degrees.

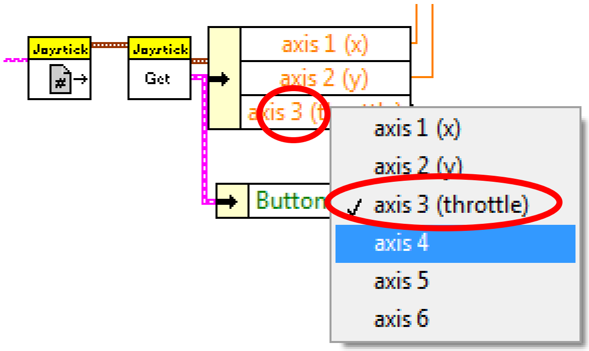
Therefore, the formula is *(Throttle User Input + 1) * 85 = Servo Angle*

17. To access math functions in LabVIEW, right-click on the block diagram and click the **Numeric** palette.



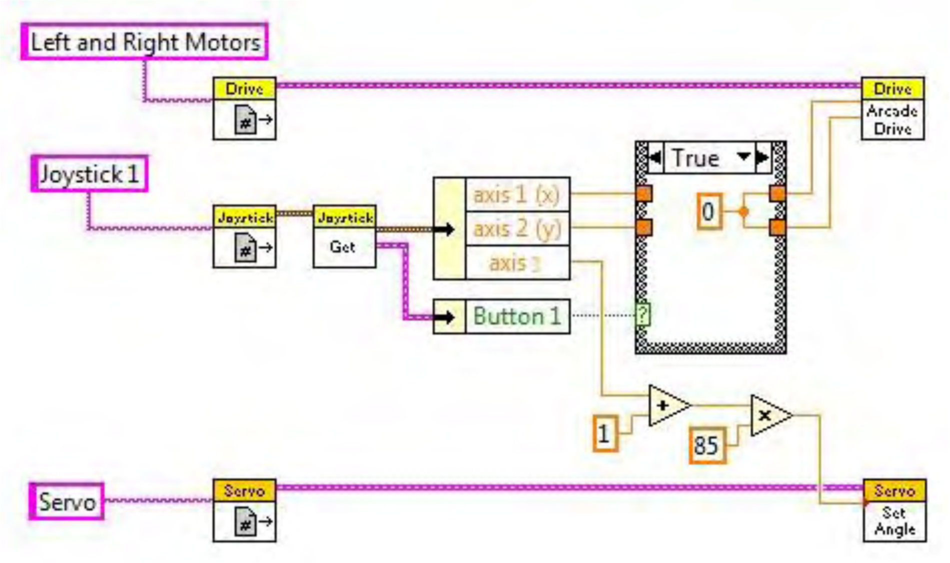
18. Since the Add and Multiply functions are used in the equation, insert them into the block diagram from the Numeric palette.

19. Expand the axis cluster Unbundle by Name function by dragging it down. Then click on the new block and select axis 3 (throttle).



20. Insert the **ServoRefNumRegistry Get VI** to get the servo reference and insert a **ServoSetAngle VI** to change the servo motor position. You can find both of these VIs on the Functions»WPI Robotics Library»Actuators»Servo palette, as in steps 2 and 4.

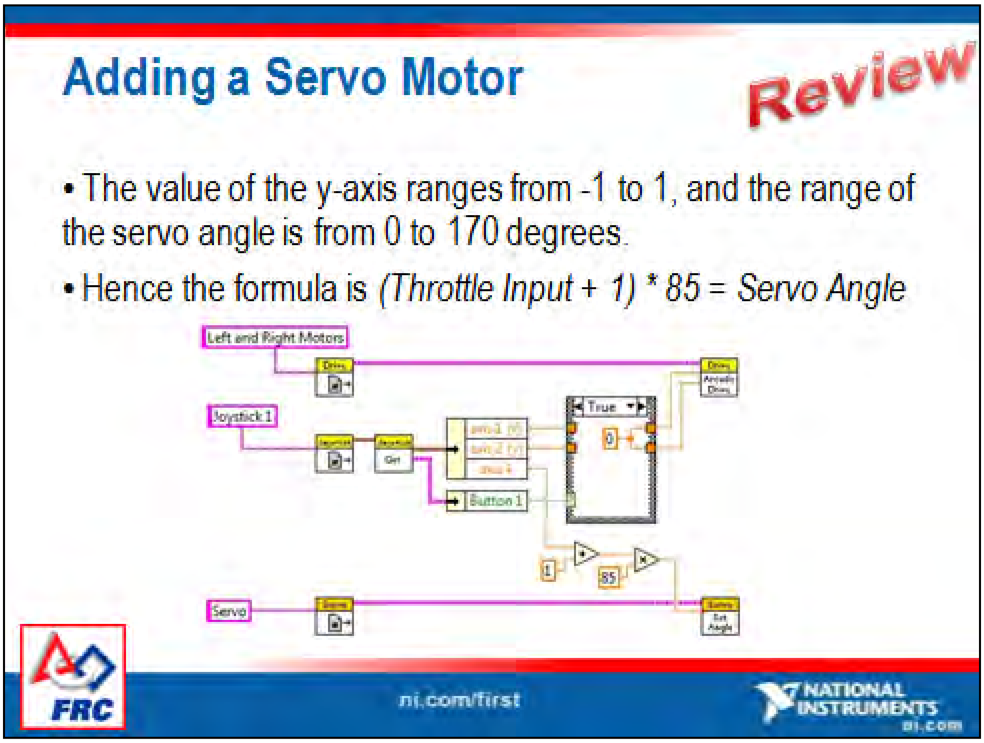
21. Wire the ServoRefNumRegistry Get and ServoSetAngle VIs together. Then wire all the math functions, numeric and string constants as shown below. NOTE: Remember you can create constants quickly by hovering the mouse over the input terminal and right-clicking and selecting **Create**, and then clicking **Constant.**



22. Close and save Teleop VI.

23. Return to Robot Main and run it to test the code. By moving the throttle input on the joystick the servo motor should spin.

END OF FRC EXERCISE 3



Adding a Servo Motor

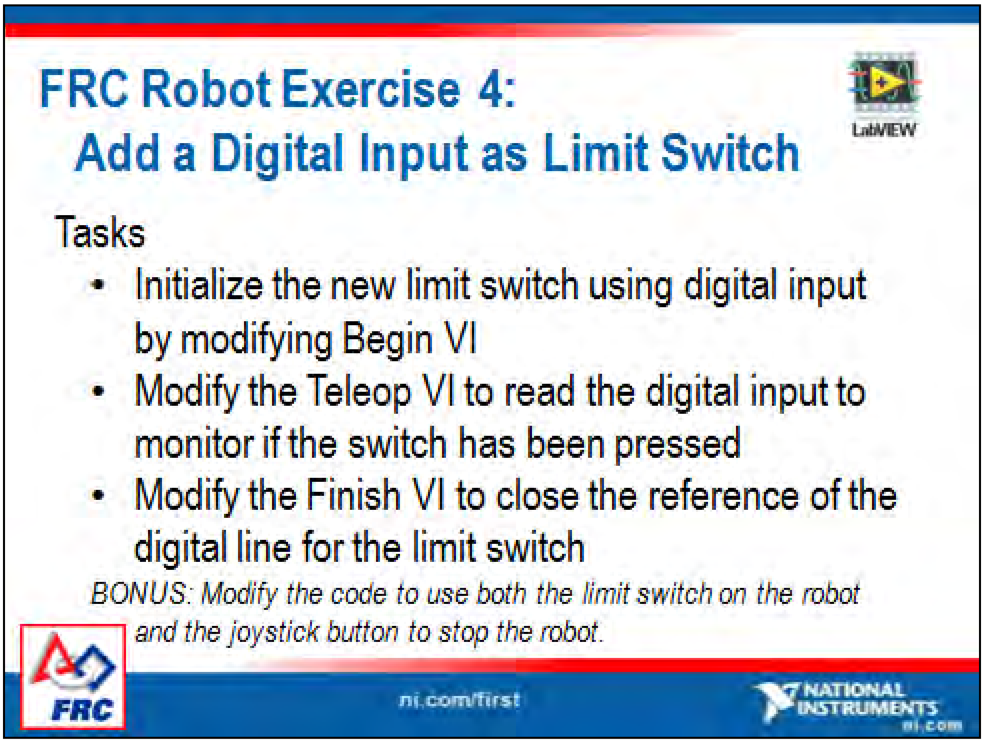
Review

- The value of the y-axis ranges from -1 to 1, and the range of the servo angle is from 0 to 170 degrees.
- Hence the formula is *(Throttle Input + 1) * 85 = Servo Angle*

FRC

ni.com/first

NATIONAL INSTRUMENTS
ni.com



FRC Robot Exercise 4: Add a Digital Input as Limit Switch

Tasks

- Initialize the new limit switch using digital input by modifying Begin VI
- Modify the Teleop VI to read the digital input to monitor if the switch has been pressed
- Modify the Finish VI to close the reference of the digital line for the limit switch

BONUS: Modify the code to use both the limit switch on the robot and the joystick button to stop the robot.

ni.com/first

In this exercise, add a limit switch to the robot. These switches are commonly used to determine if an object (wall, ball, and so on) is pressing up against the robot. The limit switch is monitored by a digital input line. This is accomplished by initializing the new switch by modifying Begin VI, modifying the Teleop VI to read the switch, and finally modifying the Finish VI to close the reference of the new switch device.

For our robot, we will replace the joystick button used as a brake from the previous exercise with the limit switch. BONUS: Try to incorporate both the switch AND the limit switch as a brake for the robot.

FRC Robot Exercise 4: Add a Digital Input as Limit Switch

Use this exercise to add a limit switch to the robot. These switches are commonly used to determine if an object (wall, ball, and so on) is pressing up against the robot. The limit switch is monitored by a digital input line. This is accomplished by initializing the new switch by modifying Begin SubVI, modifying the Teleop SubVI to read the switch, and finally modifying the Finish SubVI to close the reference of the new switch device.

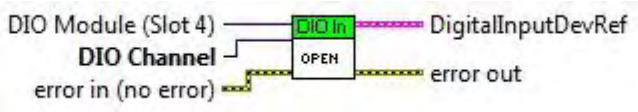
For your robot, replace the joystick button used as a brake from the previous exercise with the limit switch. **BONUS**: Try to incorporate both the switch AND the limit switch as a brake for the robot.

Digital Input Initialization

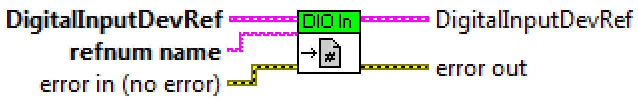
1. Open the block diagram of Begin.vi. You can find Begin.vi in the Project Explorer for the FRC project created in Exercise 1 or in the block diagram of Robot Main.vi.

Digital input initialization is similar to servo initialization from the previous exercise.

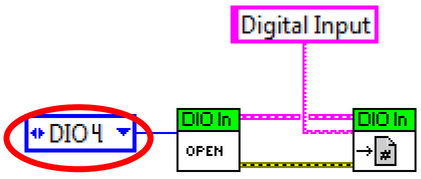
2. Insert the **DigitalInput Open** VI to configure the digital input. To insert it, right-click on the block diagram, click **WPI Robotics Library**, click **IO** palette, and then click **Digital Input** palette. Select **DigitalInputOpen** VI and place it on the block diagram.



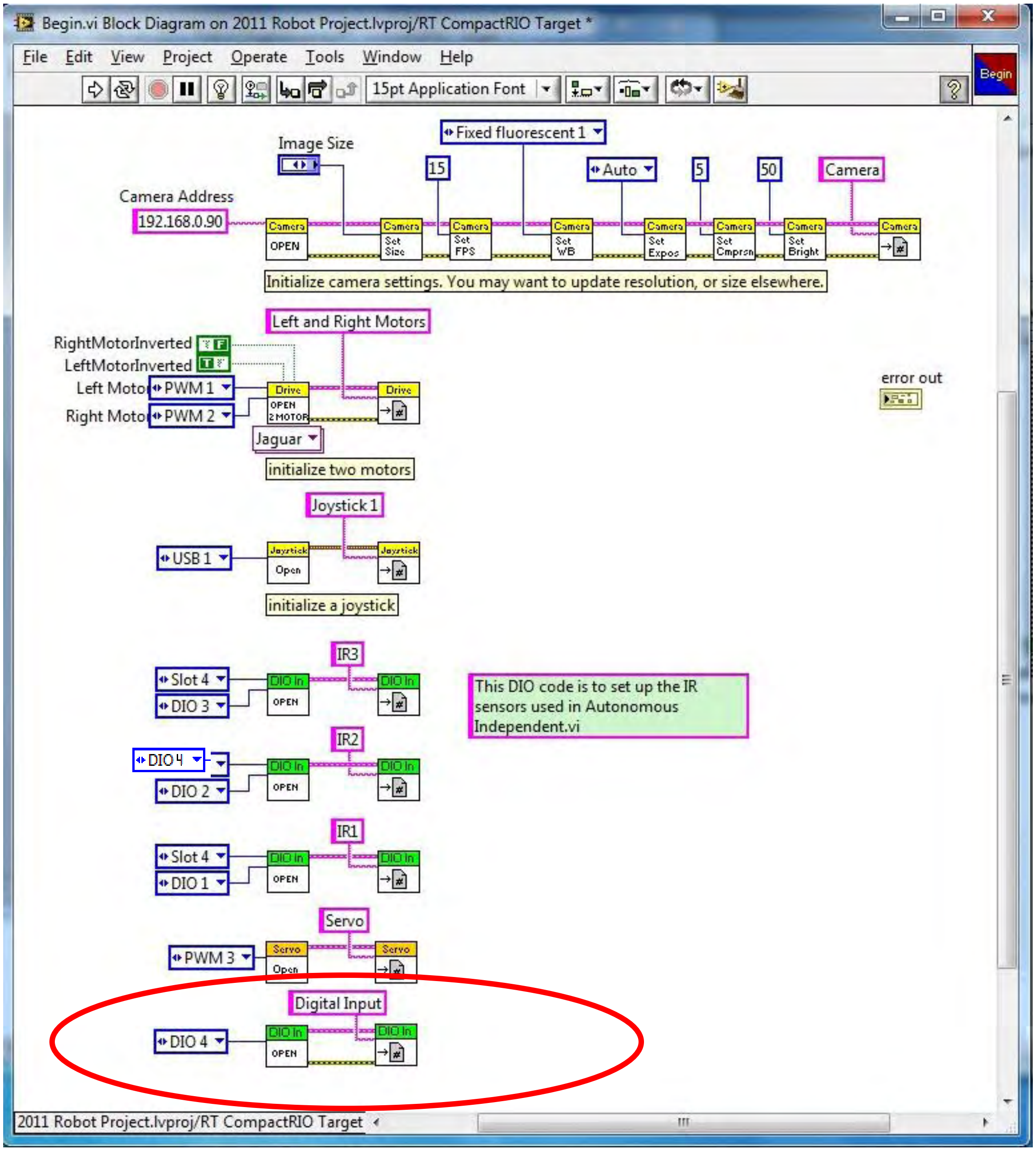
3. Place the **DigitalInputRefNum Registry Set** VI to register the digital input.



4. Create **DIO Channel** and **refnum name** constants and wire them as shown below. Select DIO 4 from the pull-down list. (DIO 1-3 are reserved for Infrared Sensors that can be used in the autonomous mode.)



Wire together the reference and error clusters. The Begin.VI block diagram should resemble the image below, with the added code circled.



Begin.vi Block Diagram on 2011 Robot Project.lvproj/RT CompactRIO Target *

Initialize camera settings. You may want to update resolution, or size elsewhere.

initialize two motors

initialize a joystick

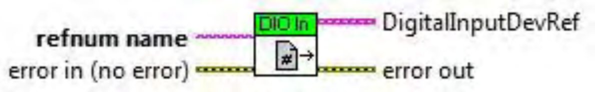
This DIO code is to set up the IR sensors used in Autonomous Independent.vi

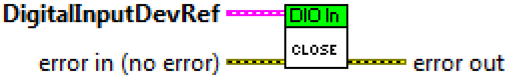
Digital Input Reference Close

5. Open the **Finish VI** block diagram.

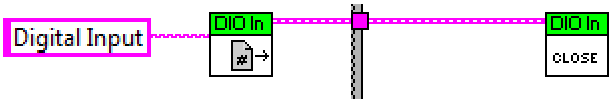
The digital input reference close is similar to the servo reference close.

6. Place the **DigitalInputRefNum Registry Get** and the **DigitalInputClose** Vis on the block diagram of Finish VI.

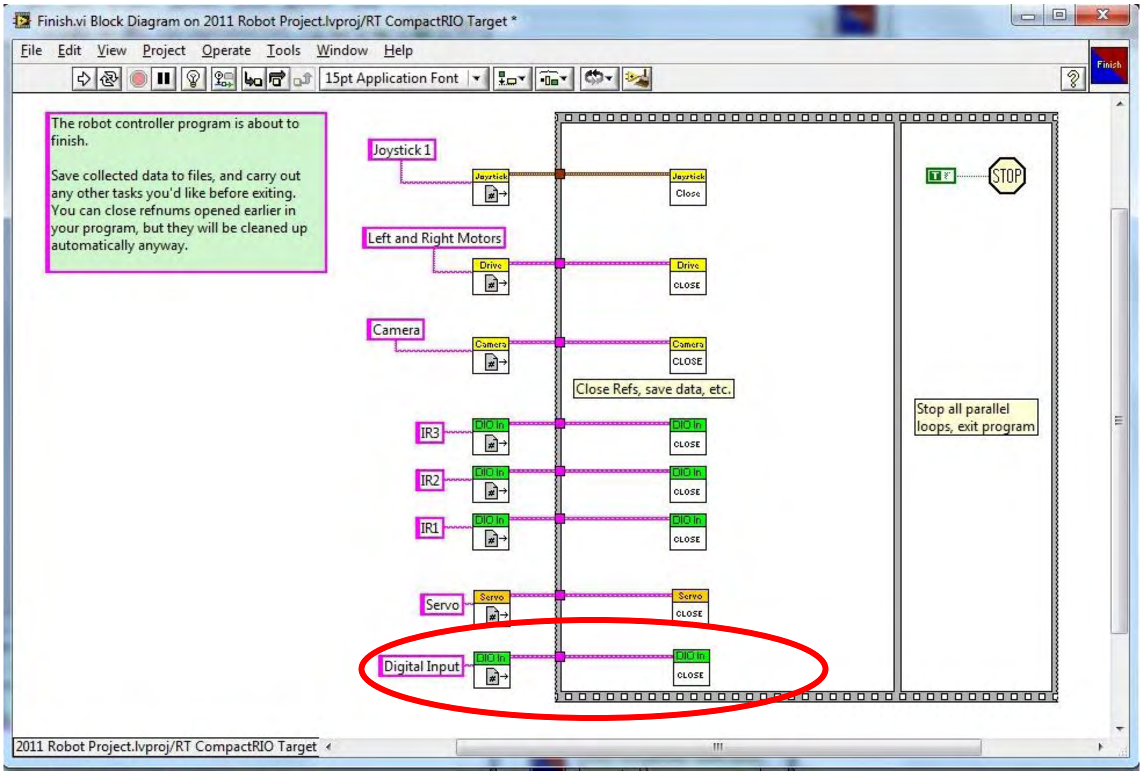




7. Wire a string constant to the Refnum Name input and then wire the reference clusters together.



The Finish.VI block diagram should resemble the image below, with the added code circled.


Finish.vi Block Diagram on 2011 Robot Project.lvproj/RT CompactRIO Target *
File
Edit
View
Project
Operate
Tools
Window
Help
15pt Application Font
Joystick 1
Left and Right Motors
Camera
IR3
IR2
IR1
Servo
Digital Input
Close Refs, save data, etc.
Stop all parallel loops, exit program
STOP
2011 Robot Project.lvproj/RT CompactRIO Target

The robot controller program is about to finish.

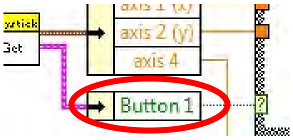
Save collected data to files, and carry out any other tasks you'd like before exiting. You can close refnums opened earlier in your program, but they will be cleaned up automatically anyway.

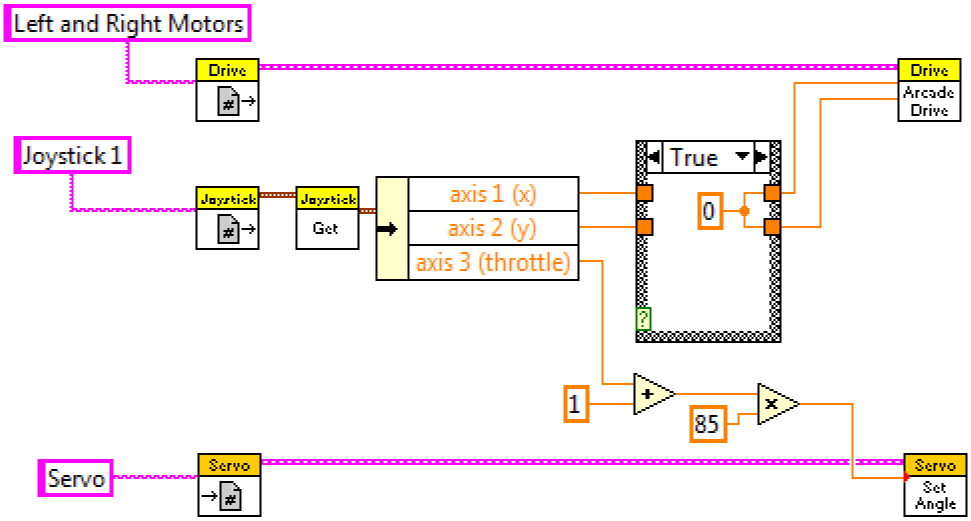
8. Close and save Finish VI.

Replace the Momentary Button With a Limit Switch

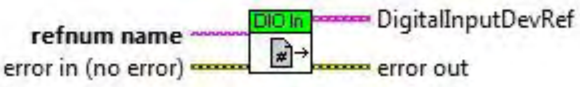
9. Open the Teleop SubVI to modify the code. You can find Teleop.vi in the Project Explorer for the FRC project created in Exercise 1 or in the “Teleop Enabled” case of the case structure from the Robot Main.vi block diagram.

10. Remove the Button cluster Unbundle by Name function. To delete it, highlight it and press delete. To remove broken wires, press **<Ctrl-B>.** The Unbundle cluster you need to delete is circled below. The code after the Button is removed is also shown below.

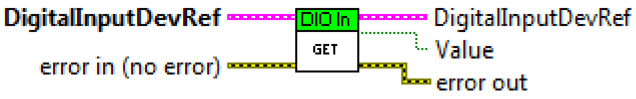




11. Place the DigitalInputRefNum Registry Get VI (may be called WPI_DigitalInputRefNum Registry Get.vi) to get the reference of the previously configured DIO in. You can find this VI under the Functions palette in the **WPI Robotics Library»IO»Digital Input** subpalette.

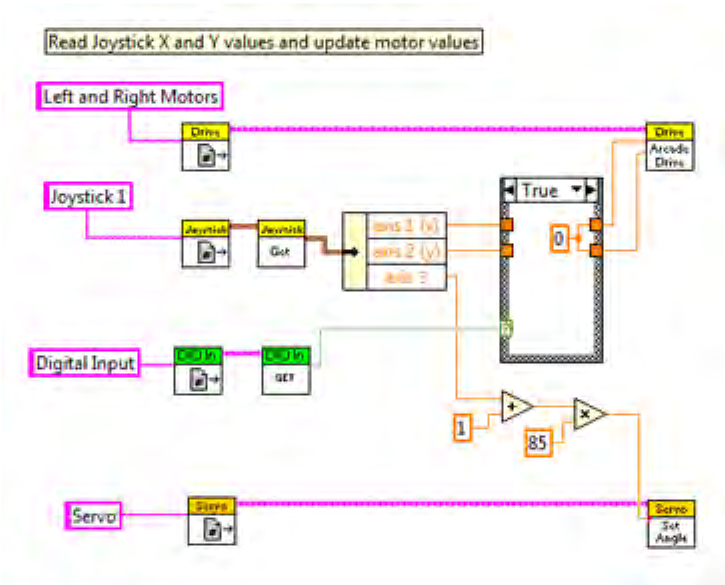


12. You also need the **DigitalInputGetValue** VI to get the input value. You can find this VI in the same digital input subpalette as the previous step. Select the **DigitalInputGetValue** VI and place it on the block diagram to the left of the case structure.



13. Create a Refnum Name constant and connect the reference cluster to the DigitalInputGetValue VI. Wire the Boolean output from DigitalInputGetValue VI to Case Structure Case Selector.

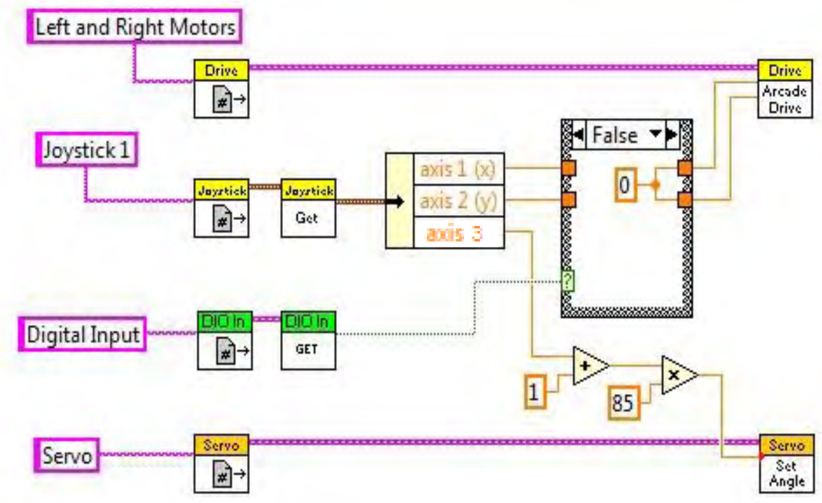
The Teleop.VI block diagram should resemble the image below.



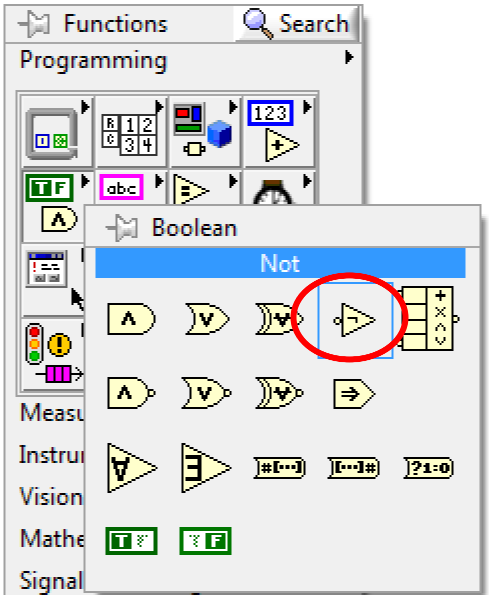
14. Reverse the logic:

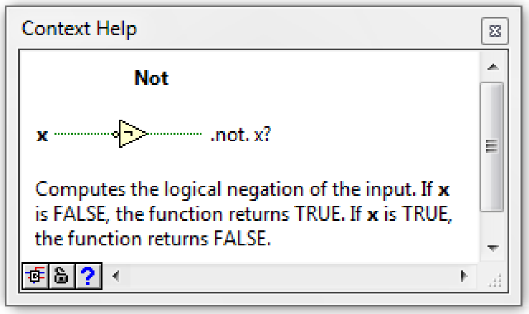
    In the case of this limit switch, when it is pressed (or closed), the output value is **False**. When the limit switch is not pressed (open), the output value is **True**. Since you want the robot to stop when the limit switch is pressed (similar to what occurred when pushing the joystick button in the previous exercise), you need to invert the logic of the limit switch (Make True ->False and False->True).

    You can choose from a few options to accomplish this. * If you plan to attempt the BONUS step, jump to Method 2.* **Method 1**: You can swap the code in the case structure by right-clicking the border of the case structure and selecting **Make this case True** or **Make this case False**. The end result is shown below. Notice the False case now forces the motors to “0.”



Method 2: Add a Not function between the **DigitalInputGetValue VI** and the case structure. You want to use this method if you plan to complete the BONUS step. You can find the Not function on the Functions palette under **Programming»Boolean.**



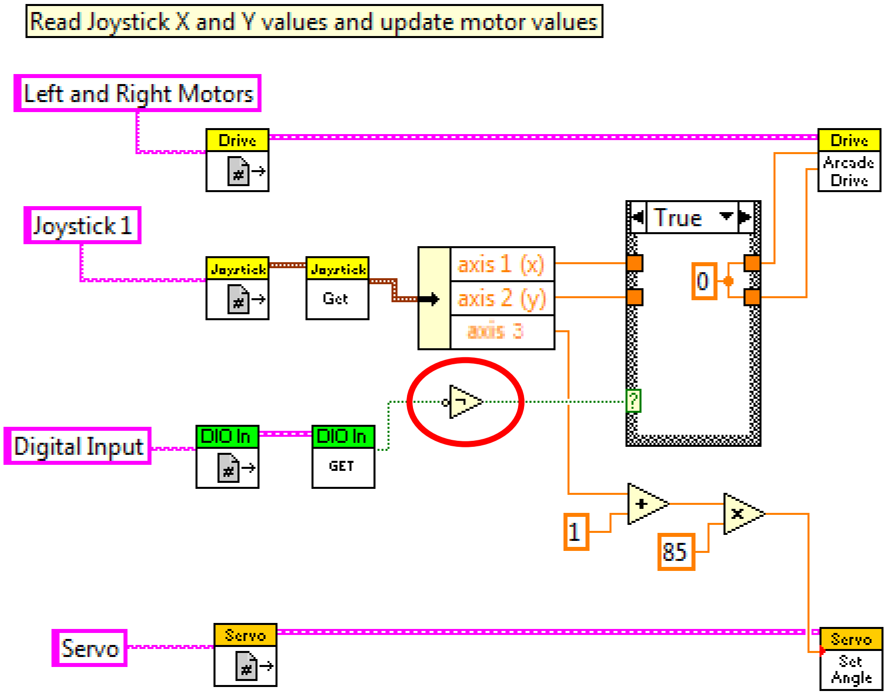

Context Help

Not

x .not. x?

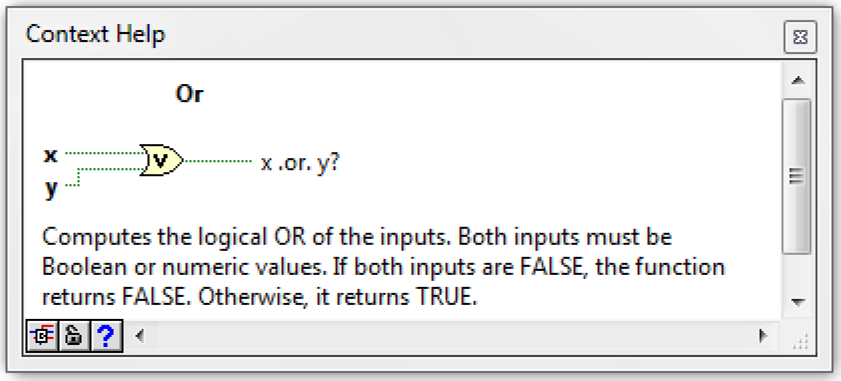
Computes the logical negation of the input. If **x** is FALSE, the function returns TRUE. If **x** is TRUE, the function returns FALSE.

The Teleop.VI block diagram is shown below.



15. BONUS: Try to incorporate both the joystick button **AND** the limit switch as a brake.

ANSWER: You can achieve this by using a Boolean function called OR.

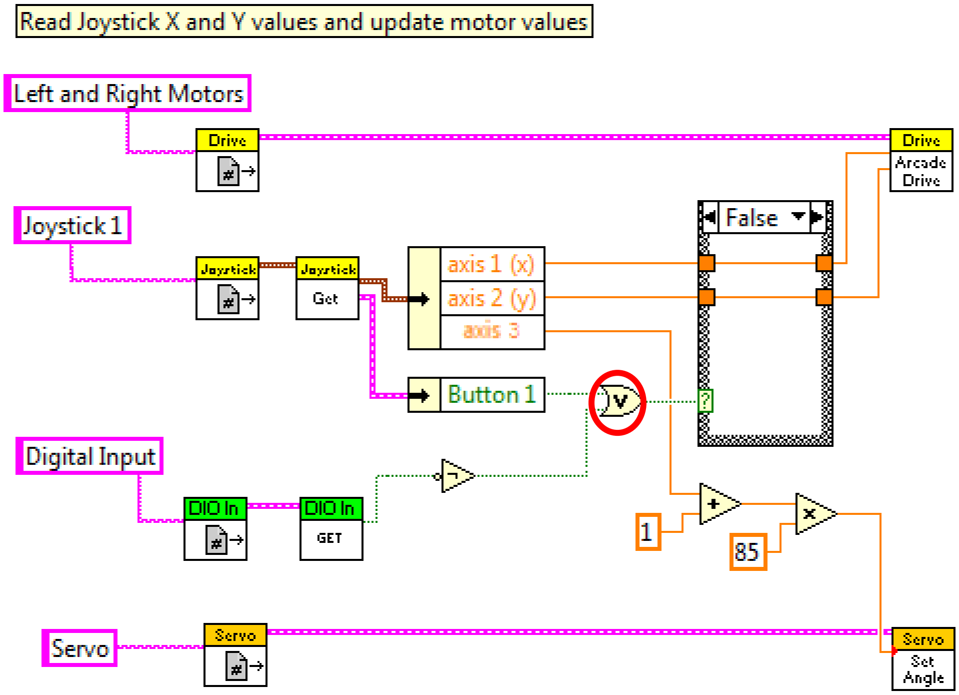


Context Help

Or

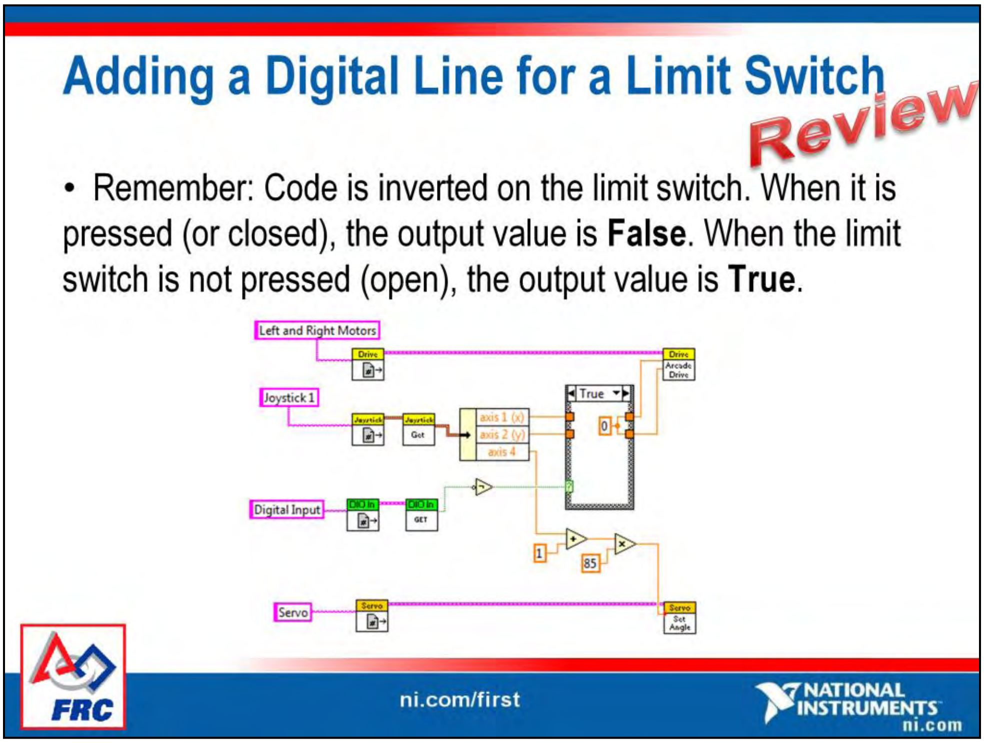
Computes the logical OR of the inputs. Both inputs must be Boolean or numeric values. If both inputs are FALSE, the function returns FALSE. Otherwise, it returns TRUE.

The Teleop.VI block diagram shown below uses both the joystick button and the limit switch as a brake for the robot.



16. Close and save Teleop VI.
17. Return to Robot Main and run it to test the code. While driving the robot you should now be able to apply a brake by depressing the limit switch (and if you completed the BONUS by also pushing Button 1 on the joystick).

END OF FRC EXERCISE 4

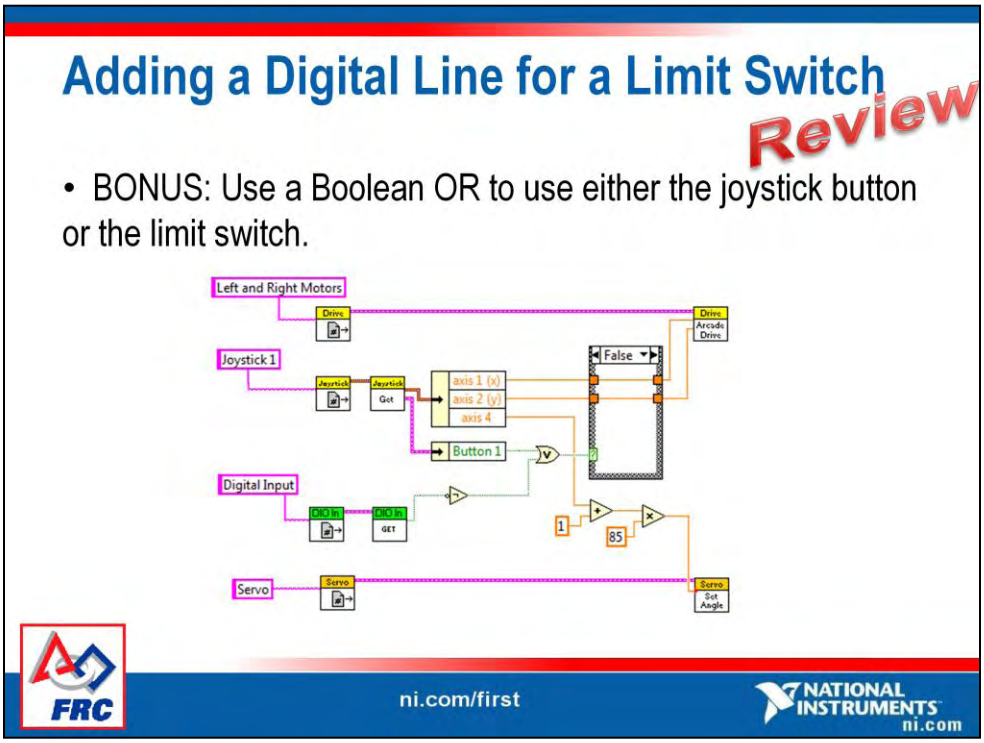


Adding a Digital Line for a Limit Switch

Review

- Remember: Code is inverted on the limit switch. When it is pressed (or closed), the output value is **False**. When the limit switch is not pressed (open), the output value is **True**.

ni.com/first



Adding a Digital Line for a Limit Switch

Review

- BONUS: Use a Boolean OR to use either the joystick button or the limit switch.

FRC

ni.com/first

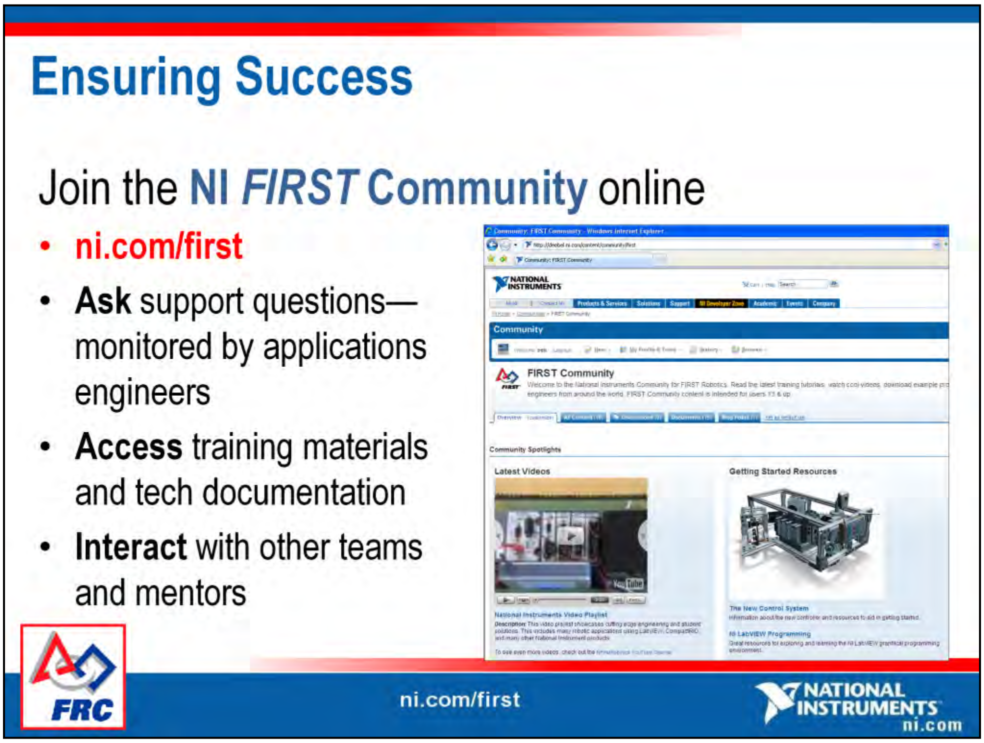
NATIONAL INSTRUMENTS

ni.com

Additional Resources and Next Steps

ni.com/first

Additional Resources and Next Steps



Ensuring Success

Join the NI *FIRST* Community online

- ni.com/first
- **Ask** support questions—monitored by applications engineers
- **Access** training materials and tech documentation
- **Interact** with other teams and mentors

Community: FIRST Community - Windows Internet Explorer

http://deebel.ni.com/content/community/first

Community

FIRST Community

Welcome to the National Instruments Community for FIRST Robotics. Read the latest training tutorials, watch cool videos, download example pro engineers from around the world. FIRST Community content is intended for users 13 & up.

Community Spotlights

Latest Videos

National Instruments Video Playlist

Description: This video playlist showcases cutting edge engineering and student solutions. This includes many robotic applications using LabVIEW, CompactRIO, and many other National Instrument products.

To see even more videos, check out the NI FIRST Robotics YouTube Channel

Getting Started Resources

The New Control System

Information about the new controller and resources to aid in getting started.

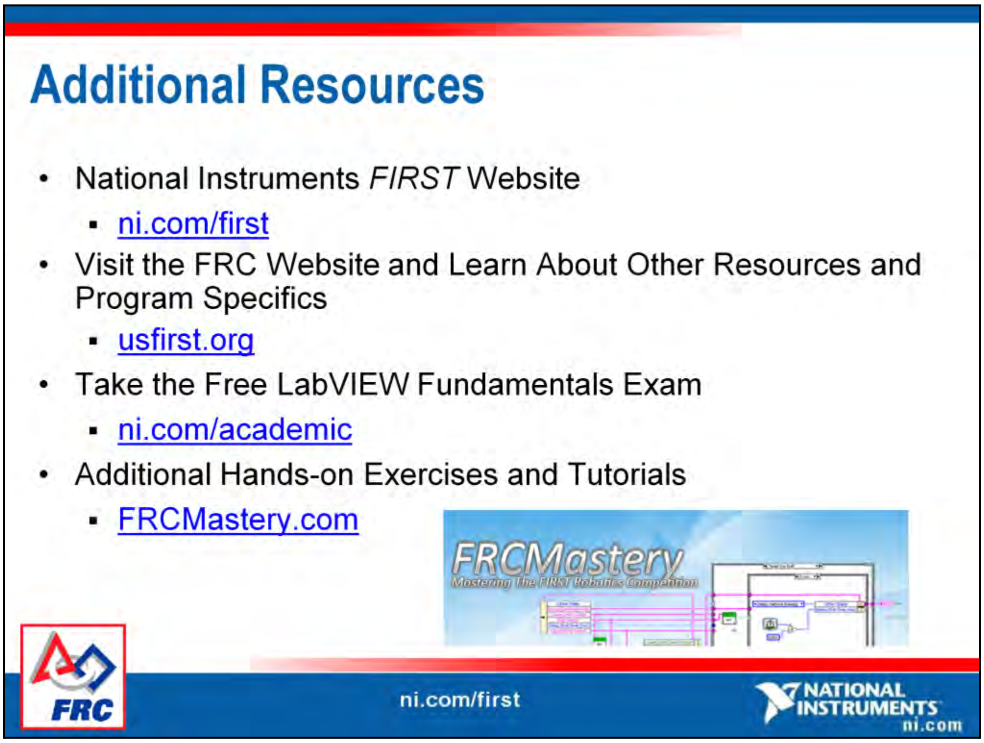
NI LabVIEW Programming

Great resources for exploring and learning the NI LabVIEW graphical programming environment.

ni.com/first

NATIONAL INSTRUMENTS ni.com

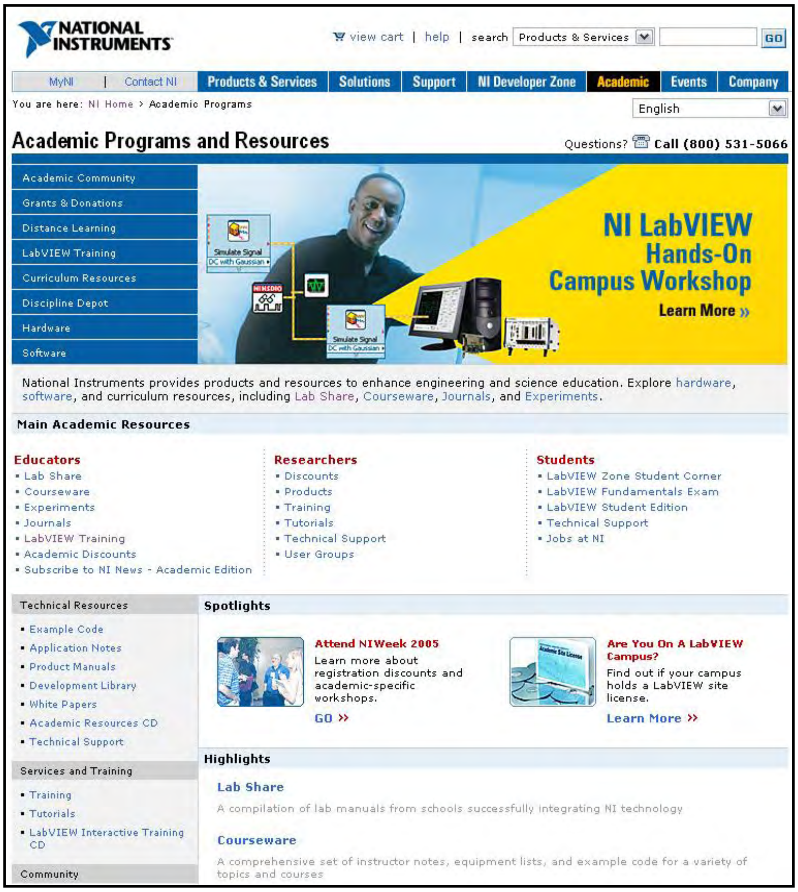
The ni.com/first community is the place to go for LabVIEW for FRC and cRIO-FRC support, specifications, training, and so on.



Additional Resources

- National Instruments *FIRST* Website
  - ni.com/first
- Visit the FRC Website and Learn About Other Resources and Program Specifics
  - usfirst.org
- Take the Free LabVIEW Fundamentals Exam
  - ni.com/academic
- Additional Hands-on Exercises and Tutorials
  - FRCMastery.com

FRCMastery
Mastering The FIRST Robotics Competition
FRC
ni.com/first
NATIONAL INSTRUMENTS
ni.com



You are here: NI Home > Academic Programs

Academic Programs and Resources

Questions? ☎ Call (800) 531-5066

National Instruments provides products and resources to enhance engineering and science education. Explore hardware, software, and curriculum resources, including Lab Share, Courseware, Journals, and Experiments.

Main Academic Resources

Educators
- Lab Share
- Courseware
- Experiments
- Journals
- LabVIEW Training
- Academic Discounts
- Subscribe to NI News - Academic Edition

Researchers
- Discounts
- Products
- Training
- Tutorials
- Technical Support
- User Groups

Students
- LabVIEW Zone Student Corner
- LabVIEW Fundamentals Exam
- LabVIEW Student Edition
- Technical Support
- Jobs at NI

Technical Resources
- Example Code
- Application Notes
- Product Manuals
- Development Library
- White Papers
- Academic Resources CD
- Technical Support

Services and Training
- Training
- Tutorials
- LabVIEW Interactive Training CD

Community

Spotlights

Attend NIWeek 2005
Learn more about registration discounts and academic-specific workshops.
GO »

Are You On A LabVIEW Campus?
Find out if your campus holds a LabVIEW site license.
Learn More »

Highlights

Lab Share
A compilation of lab manuals from schools successfully integrating NI technology

Courseware
A comprehensive set of instructor notes, equipment lists, and example code for a variety of topics and courses